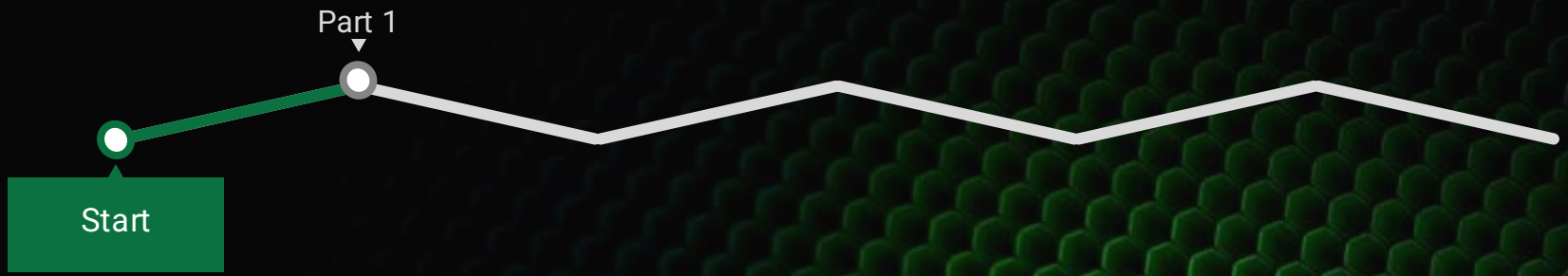
Indian Institute of Space Science and Technology, Thiruvananthapuram

Efficient Self-Supervised Neural Architecture Search

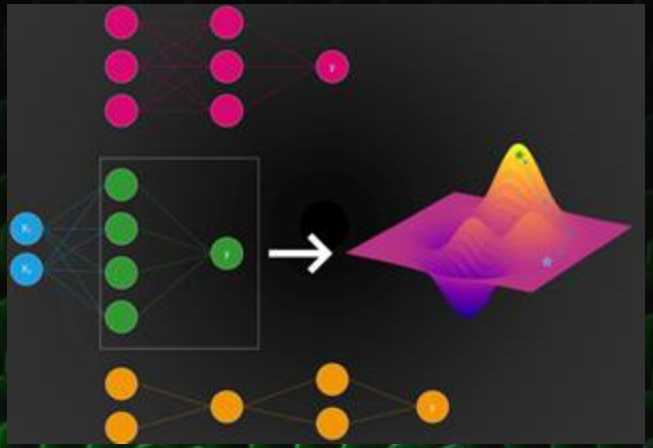
AV490 Deep Learning for Computational Data Science Course Project

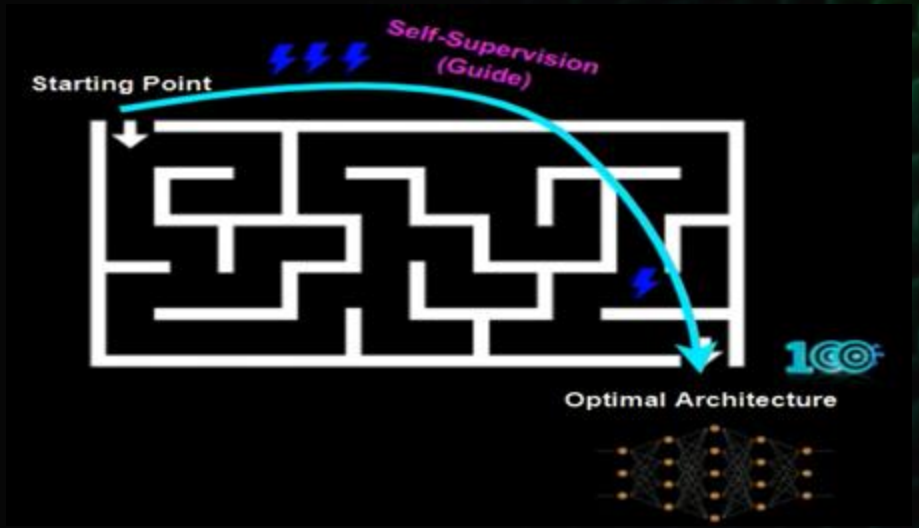


Introduction and Overview

Objective & Motivation

➔ Automate the domain knowledge intensive handcrafting of neural architecture





➔ Some of the desirable features are :

- Memory, Compute and Time Efficient Search
- “Expert-free” optimal architecture and operation selection
- Self Supervision for guiding effective architecture exploration

Types of Learning

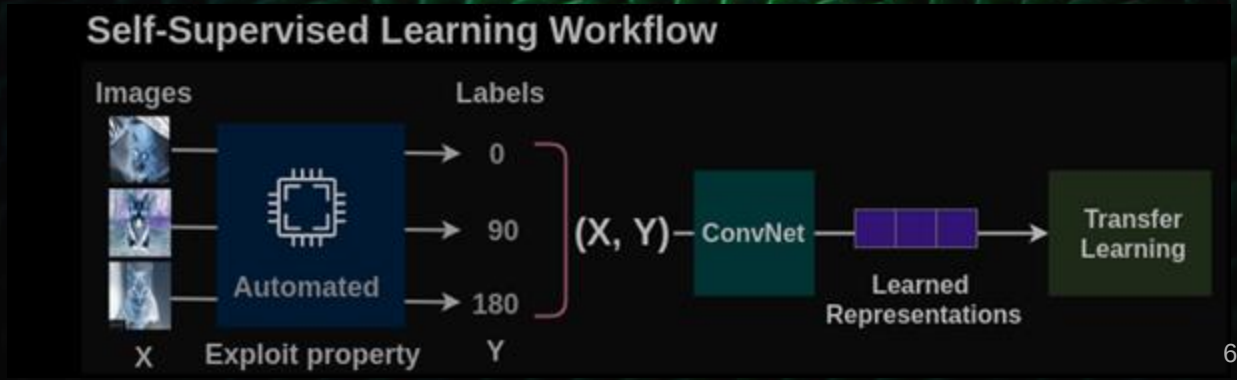
- ➔ Supervised - Learning is based upon labeled data
- ➔ Unsupervised - Learning is based upon unlabeled data
- ➔ Self-Supervised - Learning is based upon pseudo-labeled data
- ➔ Reinforcement Learning - Reward-based learning

Self-Supervised Learning

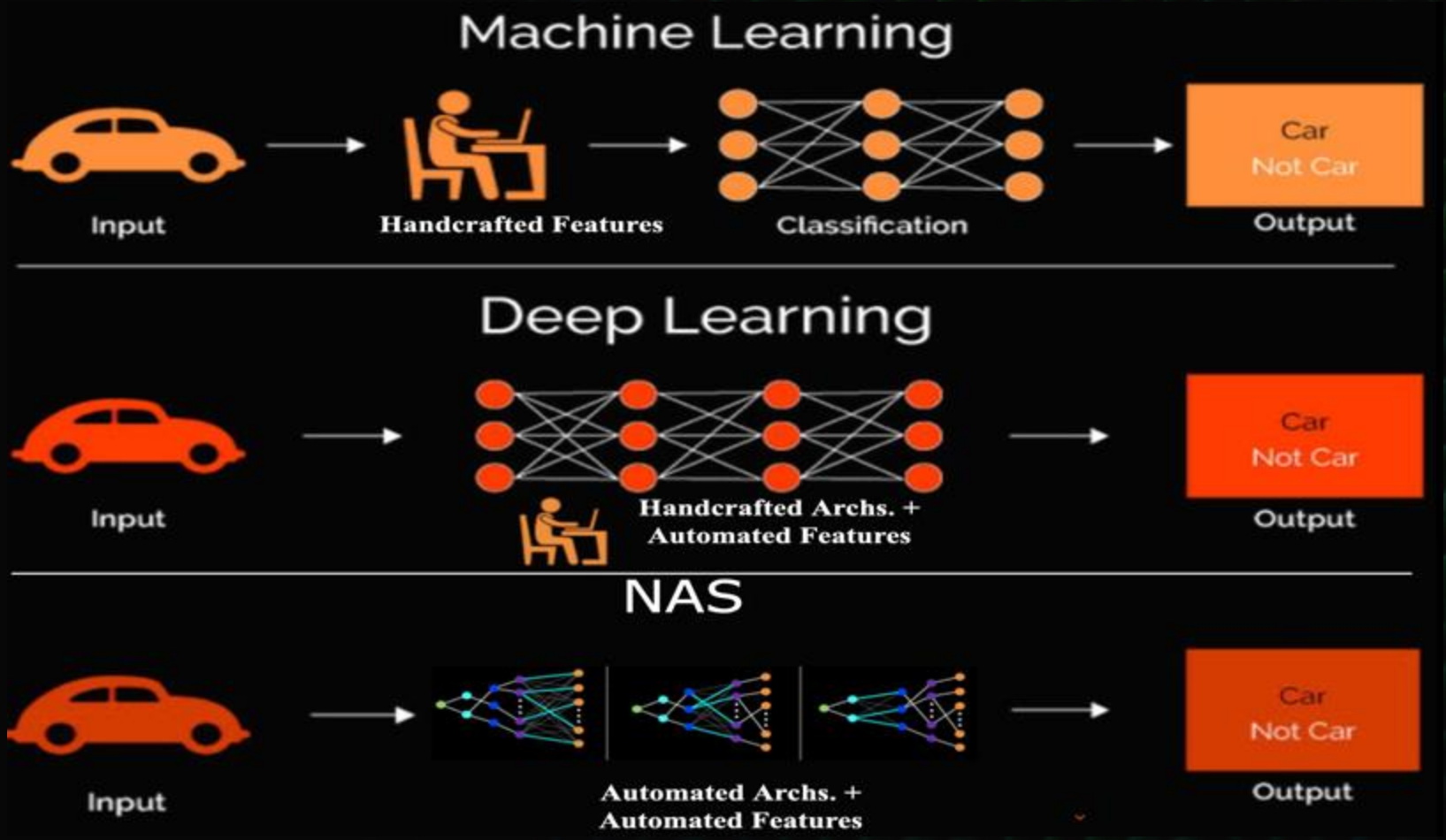
- ➔ Intermediate between Supervised and Unsupervised learning
- ➔ Self-supervised learning more closely imitates the way humans learn to classify objects
- ➔ Learns from unlabeled sample data in two steps
  - ◆ Learn pseudo-labels from an unlabelled data
  - ◆ Employ a supervised learning based on this pseudo labels
- ➔ Tries to create masks inside an unlabelled data to convert it into a pseudo-labelled data
- ➔ Idea is to create a wholesome latent vector representation for a data
- ➔ Examples - GANs, Auto-encoder

Advantages/Disadvantages of Self-Supervised Learning

- ➔ Advantages
  - ◆ No labels are required for learning
  - ◆ The network can learn any data now since labels are not necessary
  - ◆ Better feature representation are learnt
  - ◆ Non-task specific semantic meaning is captured
- ➔ Disadvantages
  - ◆ The network may take wrong pseudo-labels
  - ◆ Longer training time
  - ◆ Complex learning



Neural Architecture Search



Neural Architecture Search : Typical Challenges

➔ Most of the methods use RL or Evolutionary genetic algorithms

Memory Inefficient + **Time Consuming** + **Compute Intensive**

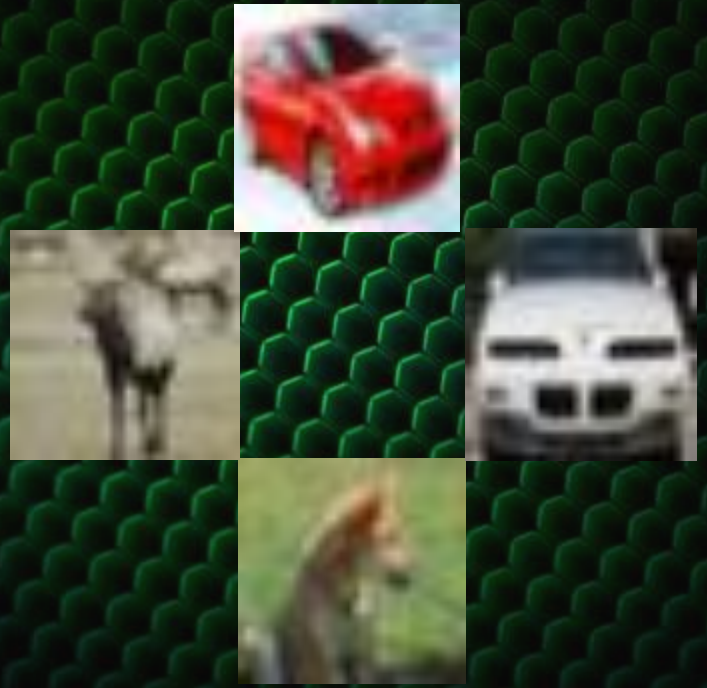
Method	GPUs	Times (days)
NAS (Zoph & Le, 2017)	800	21-28
NAS + more filters (Zoph & Le, 2017)	800	21-28
Hierarchical NAS (Liu et al., 2018)	200	1.5
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3
Progressive NAS (Liu et al., 2017)	100	1.5
NASNet-A (Zoph et al., 2018)	450	3-4
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4

Arch. Search performed on CIFAR-10

Architectural Search and Classification Dataset

CIFAR-10

- 60000 32×32 RGB images in 10 classes (6000 examples/class)
- The 10 different class labels are Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck
- Small dataset, helpful for comparing architectures



Architectural Search and Classification Dataset

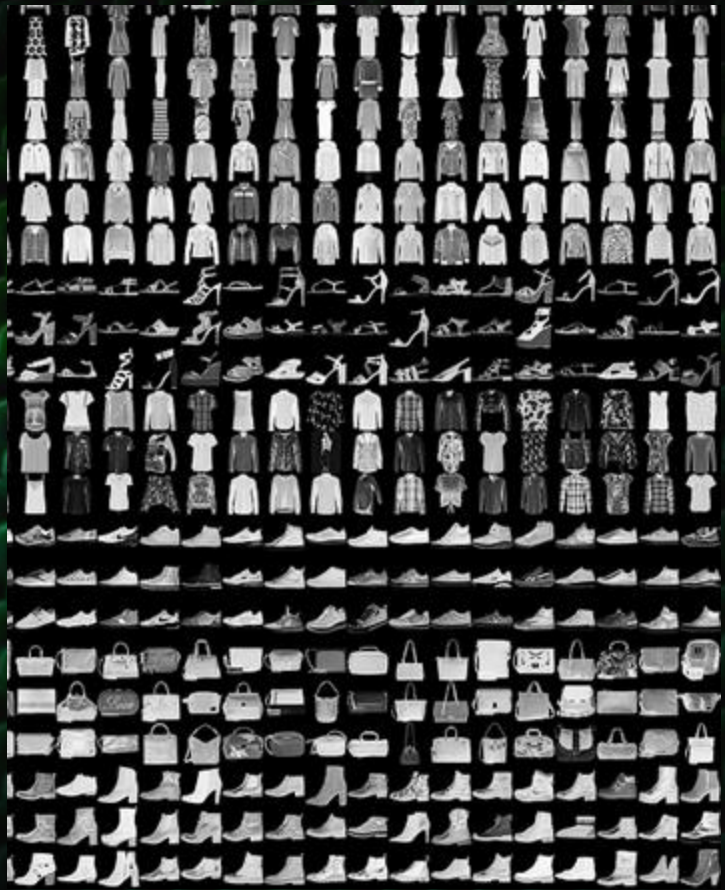
CIFAR-100

- 60000 32×32 RGB images in 100 classes (600 examples/class)
- The class labels consists of different animals, trees, foods, et cetera.
- Small dataset but with more number of classes

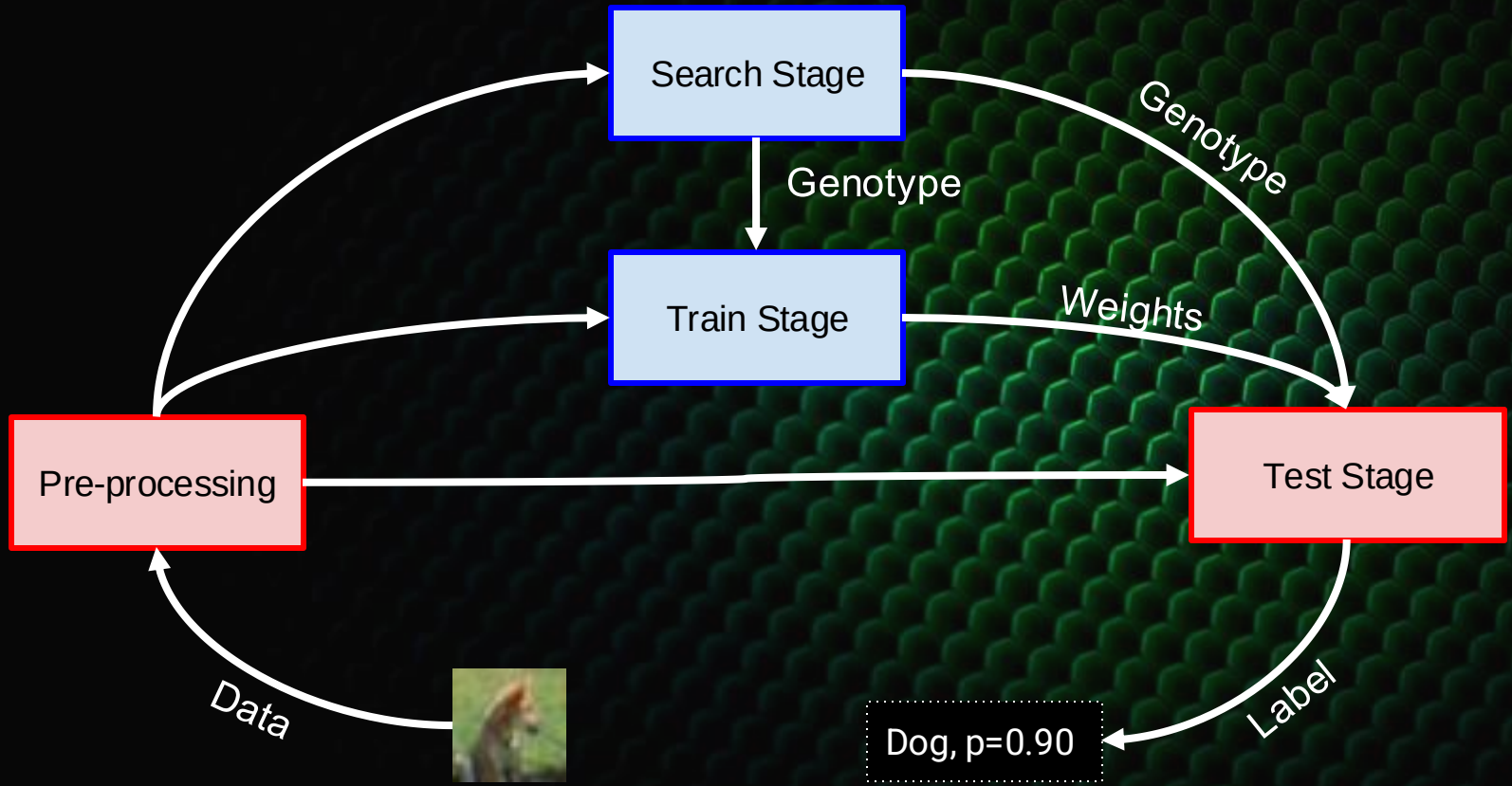
Classification Dataset

Fashion MNIST

- 70000 28×28 grayscale images in 10 classes (7000 examples/class)
- Out of this 70000, 60000 are for training and 10000 for testing
- The different class labels include T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot



Top Level Block diagram



Underlying Elemental Concepts

Differentiable Architecture Search

➔ Search of Neural Architectures based Bilevel optimization using gradients

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ \text{s.t.} \quad & w^*(\alpha) = \operatorname{argmin}_w \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

$w \rightarrow$ Network Weights

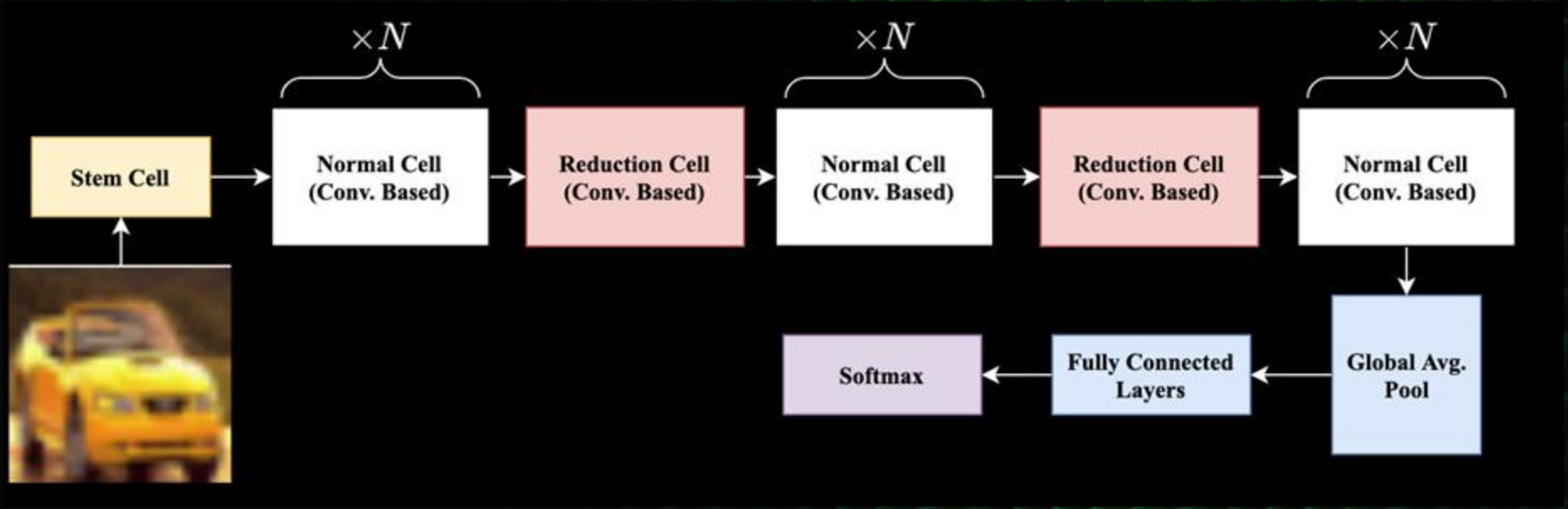
$\alpha \rightarrow$ Architecture (Ops) Weights

➔ Search space is made continuous rather than discrete set of candidate archs.

➔ Micro search for convolution operation based computational “cells”

Differentiable Architecture Search : Structure Overview

➔ Search for Normal Cell and Reduction Cell is performed



Note that N can be different during search, train and test phases !

Differentiable Architecture Search : Cell

- A cell is a directed acyclic graph consisting of an ordered sequence of ‘P’ nodes. (Here, P = 2 inputs + 4 states + 1 output = 7)

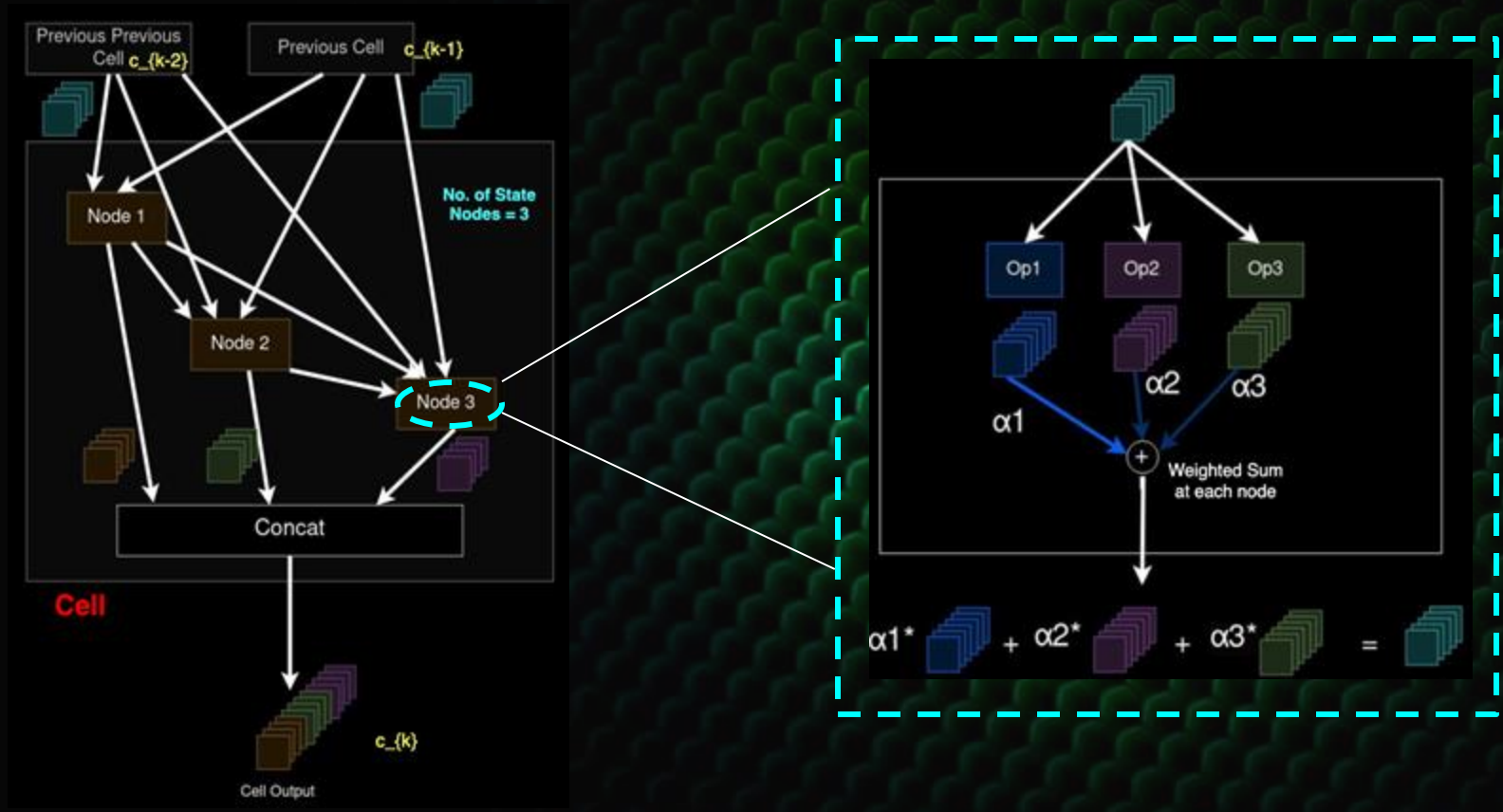
- Nodes are a set of latent feature maps. Each intermediate node is represented as :

$$x_j = \sum_{i<j} f_{i,j}\left(x_i\right)$$

- Weighted Summation happens for all “operation-outputs” at each node :

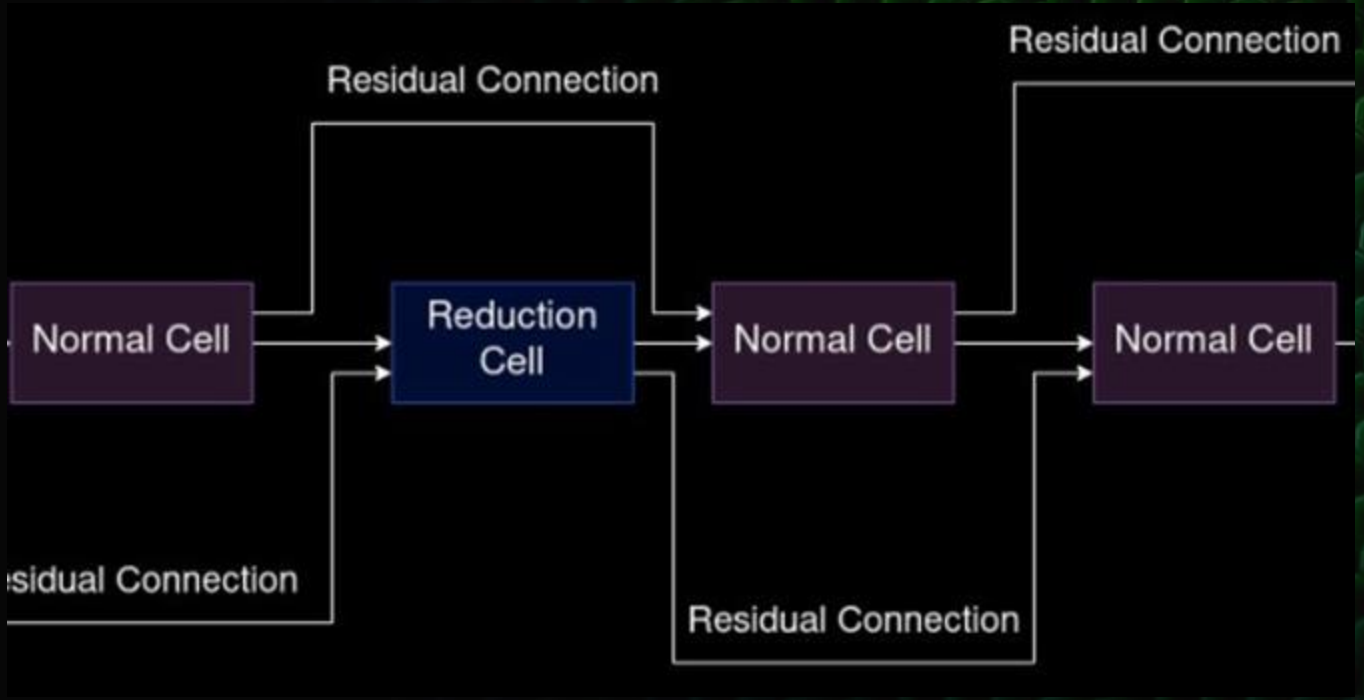
$$f_{i,j}\left(x_i\right) = \sum_{o \in \mathcal{O}_{i,j}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{(i,j)}\right)} o\left(x_i\right)$$

Differentiable Architecture Search : Cell



Differentiable Architecture Search : Cell Connections

➔ Each cell has two inputs : c_{k-1} & c_{k-2}

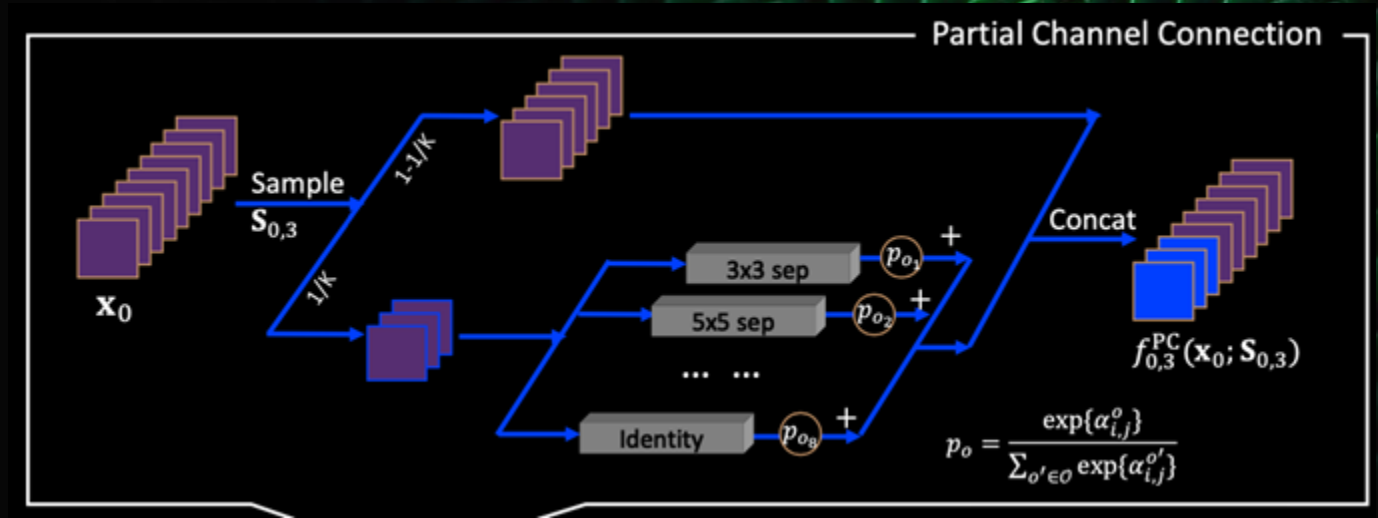


Partial Channel Connections

- ➔ Makes the search memory, computation and time efficient
- ➔ Idea : Randomly sample (1/K) of the total channels for operation selection

$$f_{i,j}^{\mathrm{PC}}(\mathbf{x}_i; \mathbf{S}_{i,j}) = \sum_{o \in \mathcal{O}} \frac{\exp\{\alpha_{i,j}^{o}\}}{\sum_{o' \in \mathcal{O}} \exp\{\alpha_{i,j}^{o'}\}} \cdot o(\mathbf{S}_{i,j} * \mathbf{x}_i) + (1 - \mathbf{S}_{i,j}) * \mathbf{x}_i \quad | \quad \mathbf{S}_{i,j} \rightarrow \text{Sampling Mask}$$

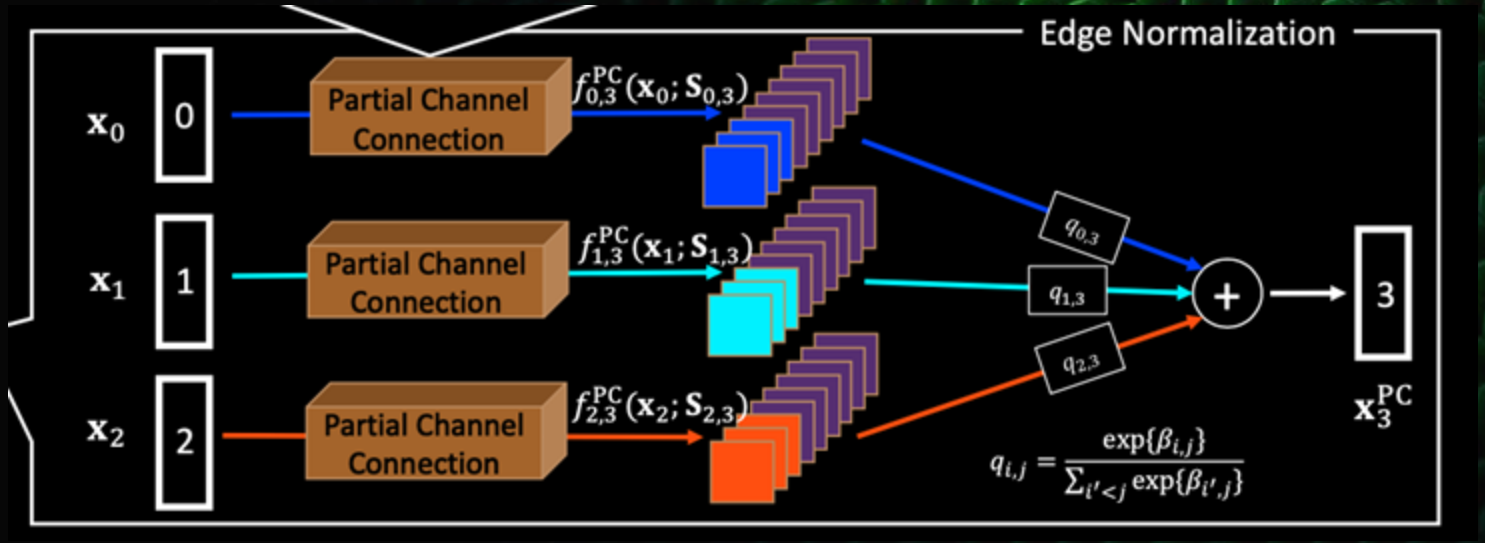
Here,
K = 4



Partial Channel Connections : Edge Normalization

- ➔ Connectivity between nodes can fluctuate due to random channel sampling
- ➔ Idea : Introduce shared, trainable parameters $\beta_{i,j}$ on each edge (i,j)

$$\mathbf{x}_j^{\mathrm{PC}} = \sum_{i<j} \frac{\exp\{\beta_{i,j}\}}{\sum_{i'<j} \exp\{\beta_{i',j}\}} \cdot f_{i,j}\left(\mathbf{x}_i\right)$$



Partial Channel Connections

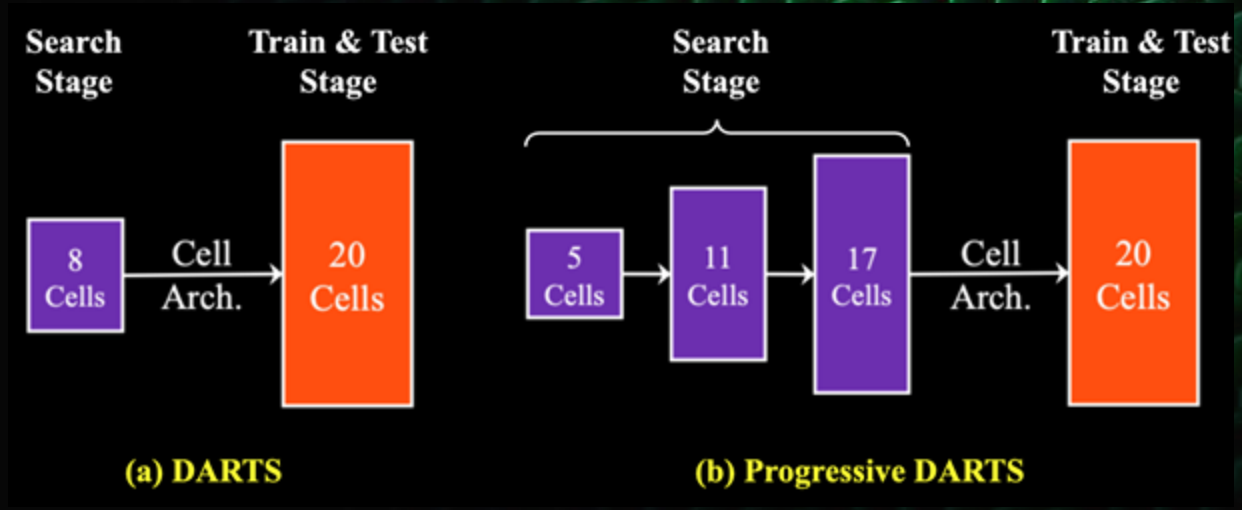
➔ Some of the typical advantages and characteristics when PC DARTS is used are :

- Bridges the Optimization Gap → Regularization effect of random channel sampling
- Time and Memory Efficiency is improved
- Batch size can increased by approx. K times → Stability of search is ensured (More training data is seen for parameter update)
- Indirectly provides regularization for weight free operations (Important for improved transfer performance)

Progressive Search

- ➔ Bridges the “Depth Gap” between search and evaluation (Certain ops. might be preferred in deep networks)

- ➔ Shallow search leads to cells with shallow connections : Degraded transfer performance on difficult datasets



Progressive Search : Search Space Approximation

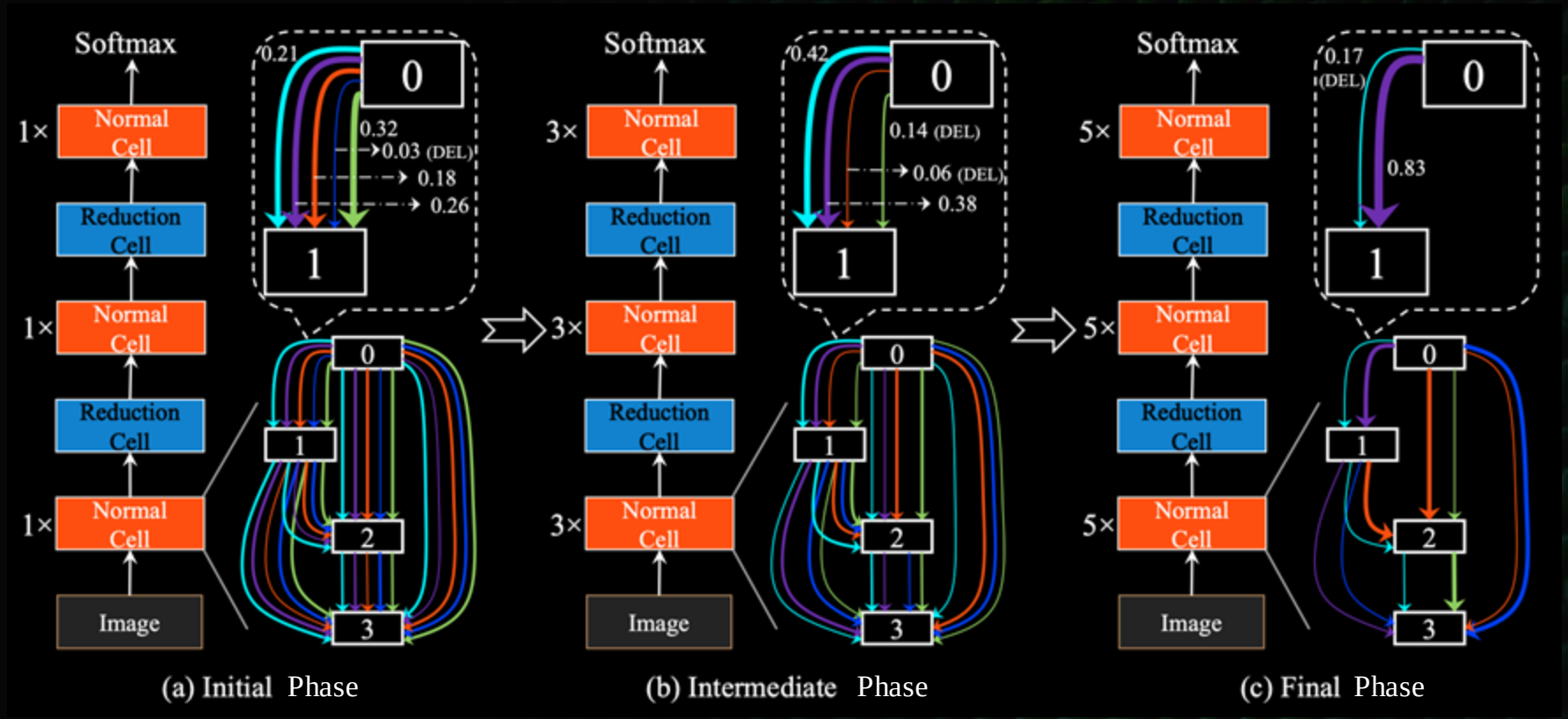
➔ Computational overhead increases linearly with the search depth, which brings issues in both time and memory : Restriction of Search space is done!

➔ Search process is split into multiple phase G_K each with L_K cells.

$$\text{As } K \uparrow \quad : \quad L_K \uparrow, \left|\mathcal{O}^k_{(i,j)}\right| \downarrow$$

➔ Idea : Dropout less important operations (assigned low weight) in the previous phase

Progressive Search : Search Space Approximation



Progressive Search : Search Space Regularization

- ➔ At each phase G_K , parameters are trained from scratch : Altered preferences for deeper architectures
- ➔ Overfitting can happen : Parameter-free skip connects are favoured more as they propagate more consistent info and leads to rapid GD on small proxy datasets
- ➔ More skipconnects : Degrades the transfer performance on difficult datasets
- ➔ Idea : Operation level Dropout for Skip connects → Partially blocked skip connections initially
- ➔ Dropout rate is decayed during training process in each search stage

Barlow Twins Mechanism

➔ Learn embeddings/latent vectors which are invariant to distortion

Take two identical networks with “slightly different” inputs

Compute the cross-correlation between the outputs

Matrix element of Cross-Correlation Matrix

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}}$$

Barlow Twins Loss

Learn as to make the cross-correlation function to identity matrix

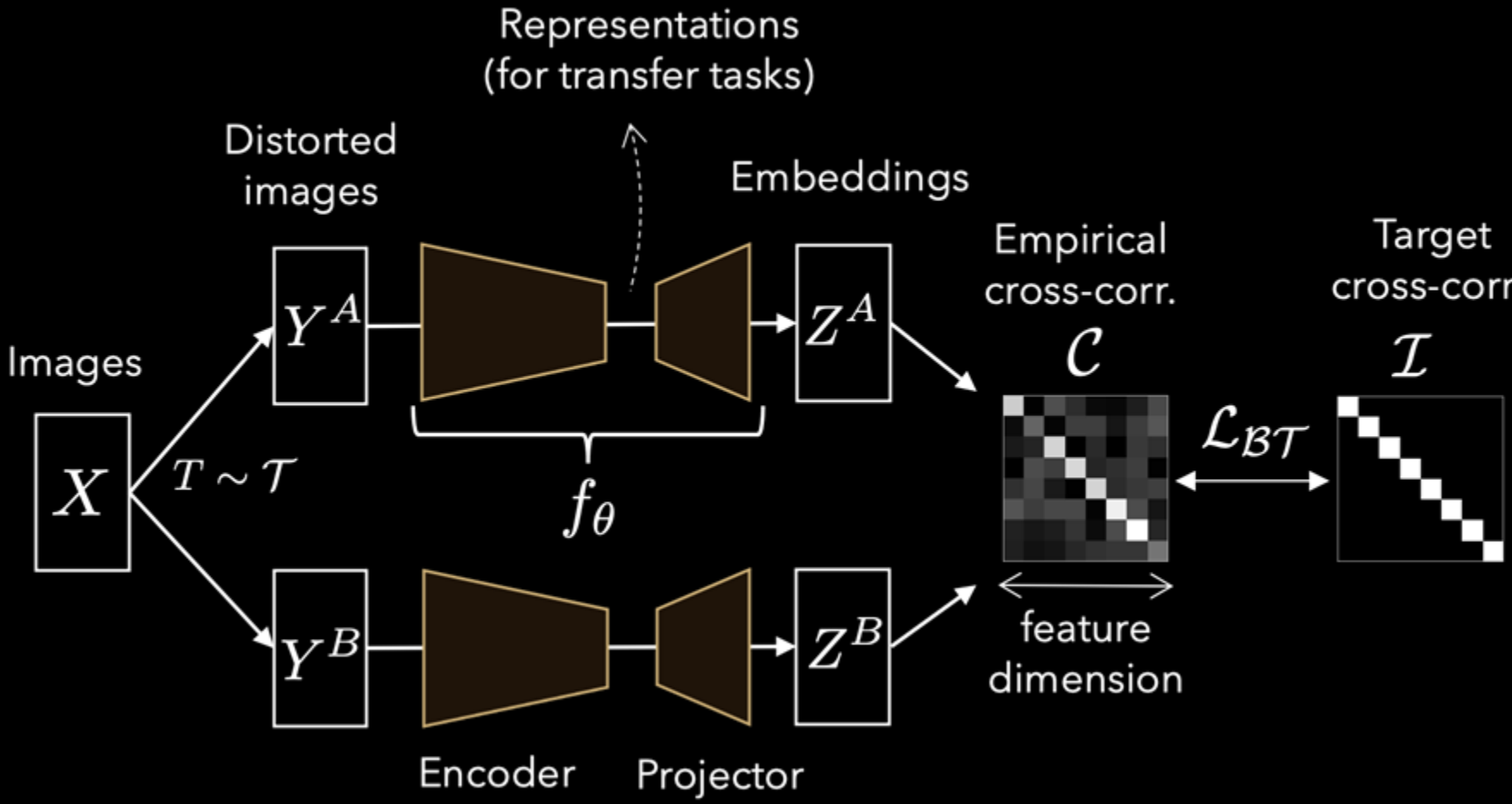
Trains the network to learn latent feature vectors

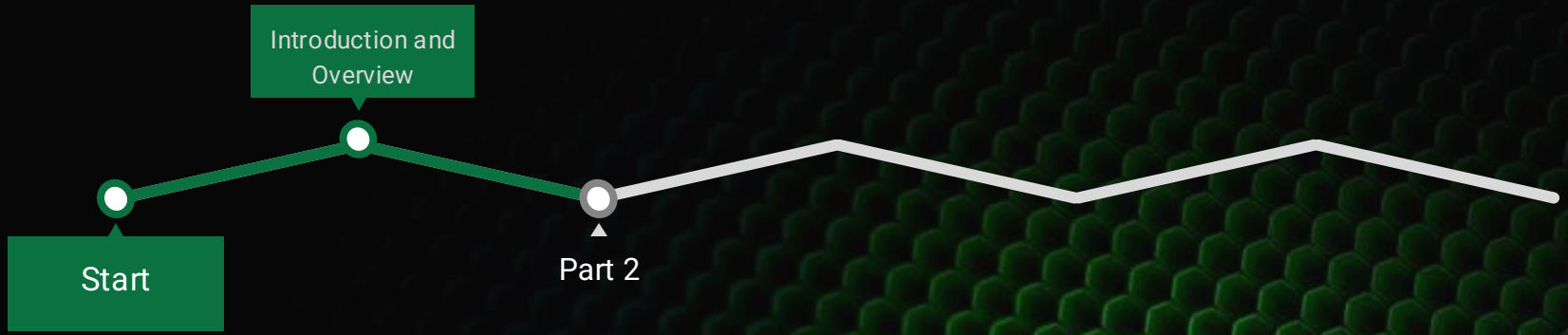
Reduces the redundancy in the embedding vectors

Ignore/generalize the distortions

Barlow Twins Loss function

$$\mathcal{L}_{BT} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{redundancy reduction term}}$$


Representations
(for transfer tasks)
Distorted
images
Embeddings
Empirical
cross-corr.
Target
cross-corr.
Images
Y^A
Z^A
$\mathcal{C}$
$\mathcal{I}$
X
$T \sim \mathcal{T}$
f_θ
$\mathcal{L}_{\mathcal{BT}}$
Y^B
Z^B
feature
dimension
Encoder
Projector



Implementation Details

General Details

- Framework Used : PyTorch (For Search, Training & Testing)

- Device Configuration :
  - For Search & Train Stages → Tesla V100 PCI-E 32 GB & 16 GB (IIST HPC),
  - For Testing & Analysis → Google Colab's Tesla GPUs and NVIDIA RTX GeForce 2060 GPU

Preprocessing Details

- PyTorch's Loader and Data Transformer was used for the following preprocessing steps, wherever required *(Specifics are mentioned in later slides)* :
    - Data Loading for Training,Validation and Testing
    - Random Crop and Random Horizontal Flip Augmentations for Search and Train Stages
    - Resizing, Converting to Tensors & Normalizing Images
- Batching → For Larger GPUs, 128-256 images/batch whereas for smaller GPUs 64-96 images/batch

Preprocessing Details : Cutout Regularization

- ➔ Process of obfuscating some square regions of input image randomly
  - ◆ Acts as a regularizer for the model
- ➔ Applied during Train Stage

Dog Image

Dog Image with Cutout

Preprocessing Details : Barlow Twins

➔ For generating the two distorted images, following transformations are applied probabilistically :

- Random Crop
- Random Horizontal Flip
- Random Color jitter
- Random Grayscale
- Random distortion
- Random Erasing

Training Configuration

- **Output Classes** → 10 Classes for CIFAR10 & FMNIST | 100 Classes for CIFAR100
- **Train Optimizer** → Adam Optimizer (lr = 6e-4, β_1=0.5,β_1=0.999) + Weight Decay
- **Search Optimizer** → SGD Optimizer (lr = 0.025) + Weight Decay + Cosine LR Scheduler
- **Checkpointing** of Models → Decrease in validation loss
- **Train - Val Split** → 50 K - 10 K
- **Auxiliary Loss Towers** → For boosting gradient Flow during Backprop.

Model Evaluation Metrics

Class-wise Metrics

$$\text{Accuracy[class]} = \frac{TP[class]+TN[class]}{TP[class]+FP[class]+FN[class]+TN[class]}$$

$$\text{Precision[class]} = \frac{TP[class]}{TP[class]+FP[class]}$$

$$\text{Recall[class]} = \frac{TP[class]}{TP[class]+FN[class]}$$

$$\text{F1 Score[class]} = \frac{2*Precision[class]*Recall[class]}{Precision[class]+Recall[class]}$$

where TP[class] - True Positives for class under consideration
FP[class] - False Positives for class under consideration
FN[class] - False Negatives for class under consideration
TN[class] - True Negatives for class under consideration

Overall Metrics **(all)**

$$\text{Accuracy} = \frac{TP}{Sum\ of\ all\ elements\ in\ Confusion\ Matrix}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{F1 Score} = \frac{2*Precision*Recall}{Precision+Recall}$$

where (considering micro-averaging method)

$$TP = \sum_{class} TP[class]$$

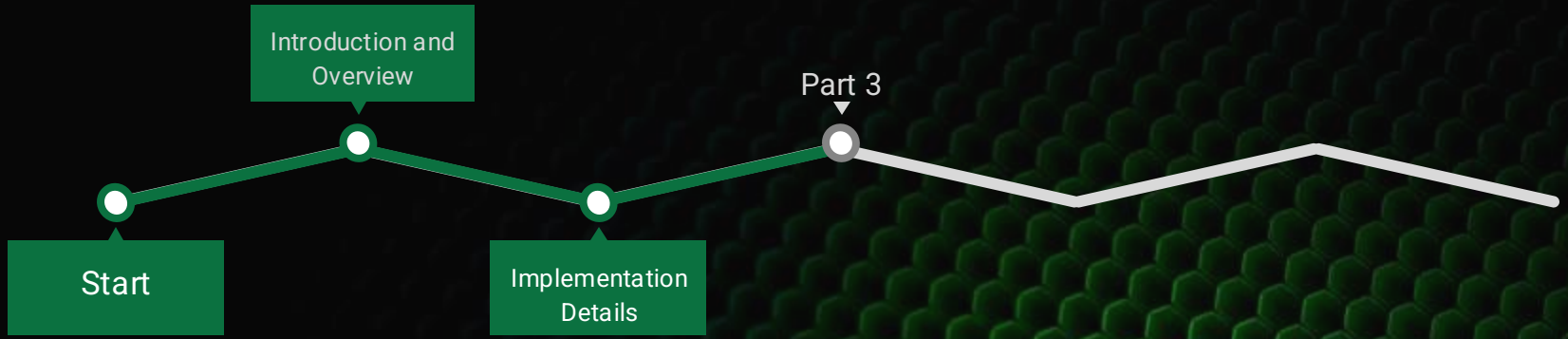
$$FP = \sum_{class} FP[class]$$

$$FN = \sum_{class} FN[class]$$

NAS Operation Space

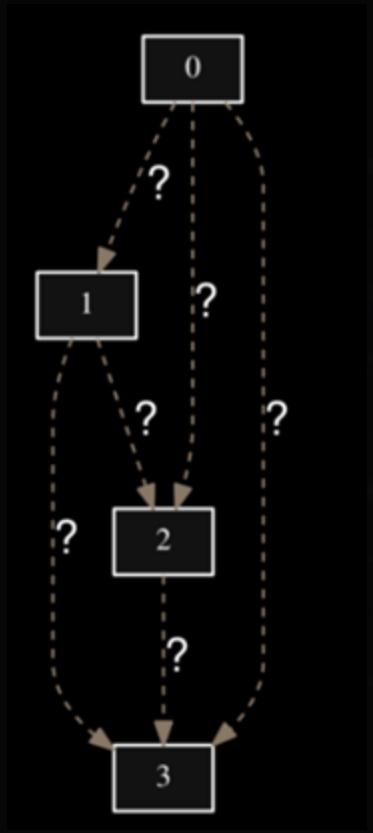
This is the list of operations we have taken in our Neural Architecture Search Space. All the operations ensure that the input and the output shape are same through zero padding.

- max_pool_3x3 - Max pooling operation with 3×3 kernel
- avg_pool_3x3 - Average pooling operation with 3×3 kernel
- skip_connect - Skip Connection
- sep_conv_3x3 - Depth-wise Separable Convolution with 3×3 kernel
- sep_conv_5x5 - Depth-wise Separable Convolution with 5×5 kernel
- dil_conv_3x3 - Dilated Convolution with 3×3 kernel
- dil_conv_5x5 - Dilated Convolution with 5×5 kernel



Approach Design

Design Considerations for Architecture Search



➔ Learning Algorithm :

◆ Non-Progressive Case

▼

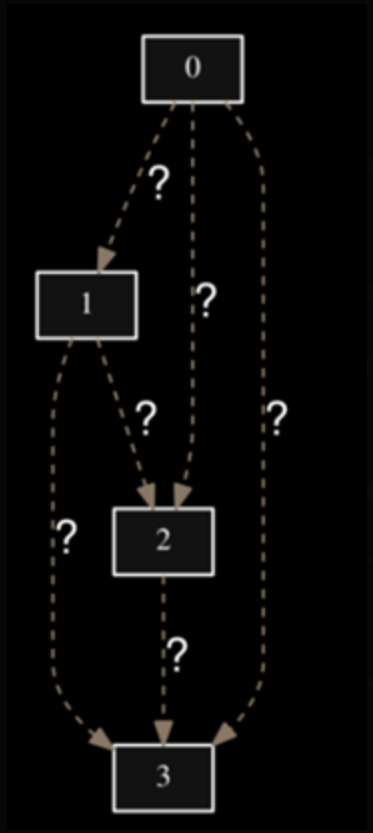
- ❖ Network Parameters optimized for complete [0,40] epochs
- ❖ Arch. Parameters are optimized for [16,40] epochs
- ❖ 8 cells arch. searched + 20 cells arch. evaluated

◆ Progressive Case

▼

- ❖ 3 phases of increased depth arch. Search
  - ➢ Phase-I → 5 cells + 8 operations
  - ➢ Phase-II → 11 cells + 5 operations
  - ➢ Phase-III → 17 cells + 3 operations
- ❖ Network Parameters optimized for complete [0,40] epochs
- ❖ Arch. Parameters are optimized for [10, 25] epochs
- ❖ 20 cells arch. evaluated

Design Considerations for Architecture Search



➔ Choosing Operations :

◆On each edge (i, j), operation with the largest $\alpha^o_{i,j}$ value is preserved

➔ Choosing Internode Connections

◆ For non-partial connection case :

Node x_j needs to be pick two links from $\{\max_o \alpha^o_{0,j}, \max_o \alpha^o_{1,j}, \ldots, \max_o \alpha^o_{j-1,j}\}$

◆ For partial-connection based search :

Node x_j needs to be pick two links from normalised values of the set $\{(\max_o \alpha^o_{0,j})\beta_{0,j}, (\max_o \alpha^o_{1,j})\beta_{1,j}, \ldots, (\max_o \alpha^o_{j-1,j})\beta_{j-1,j}\}$

How did we implement Self-Supervised NAS?

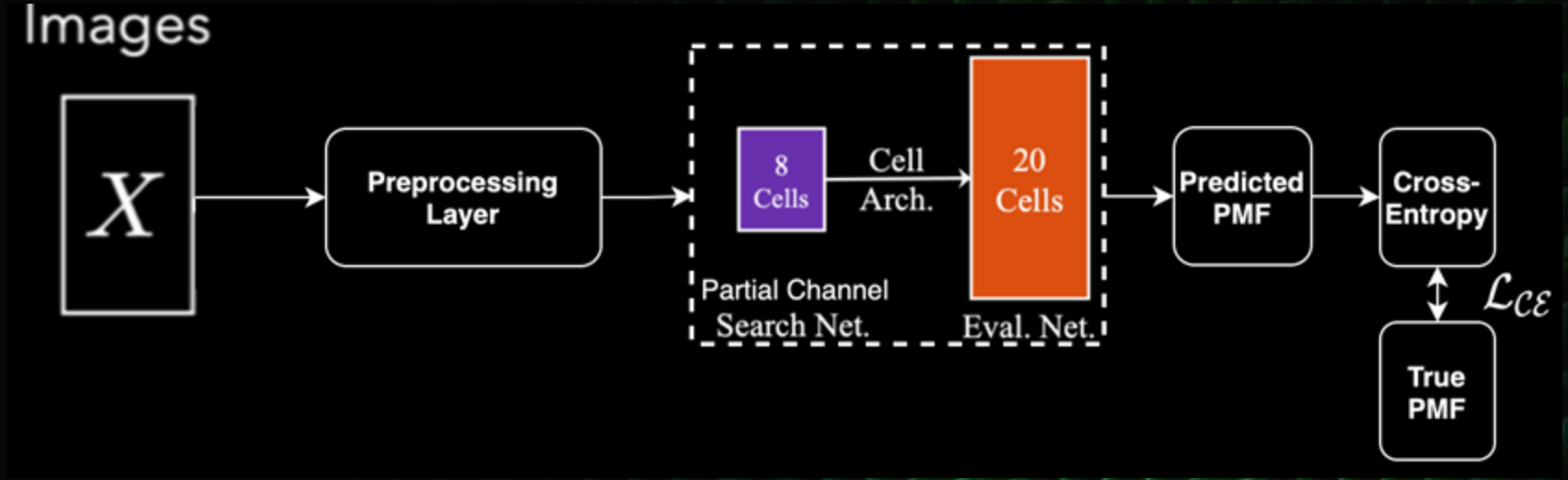
We designed a combination of loss function for supervised and self-supervised NAS.

A combination of Cross-entropy loss function and Barlow Twin Loss functions was taken for implementing self-supervised NAS.

$$\mathcal{L} = \text{ss\_factor} \times \mathcal{L}_{\mathcal{BT}} + (1 - \text{ss\_factor})\mathcal{L}_{\mathcal{CE}}$$

The ss_factor can be any real number between 0 and 1 and was chosen as 0.25, 0.5, 0.75 and 1 for various experiments

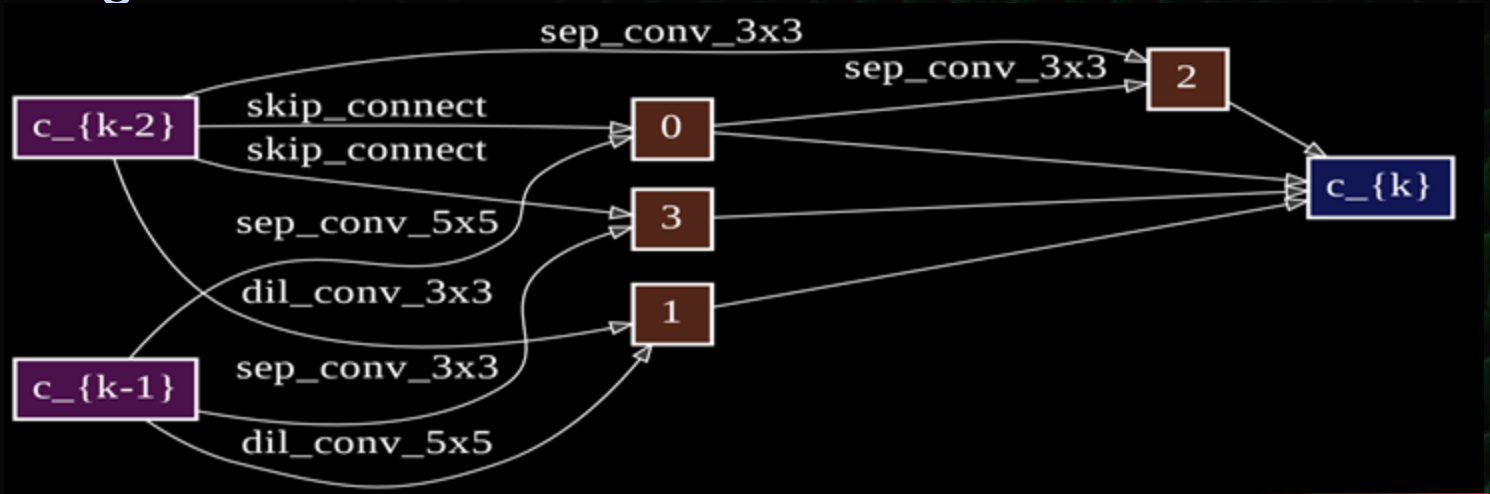
Stage 1 : Vanilla PC DARTS



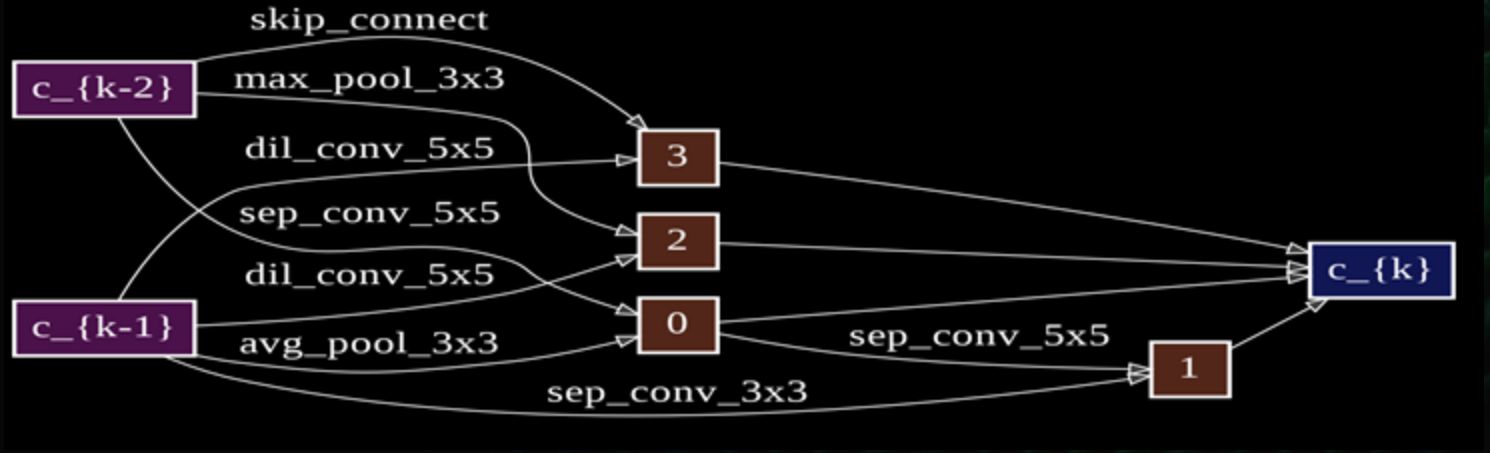
- Features are :
  - Partial Connections based Search
  - Supervised Loss for Search and Classification

Search Stage Results

Normal Cell

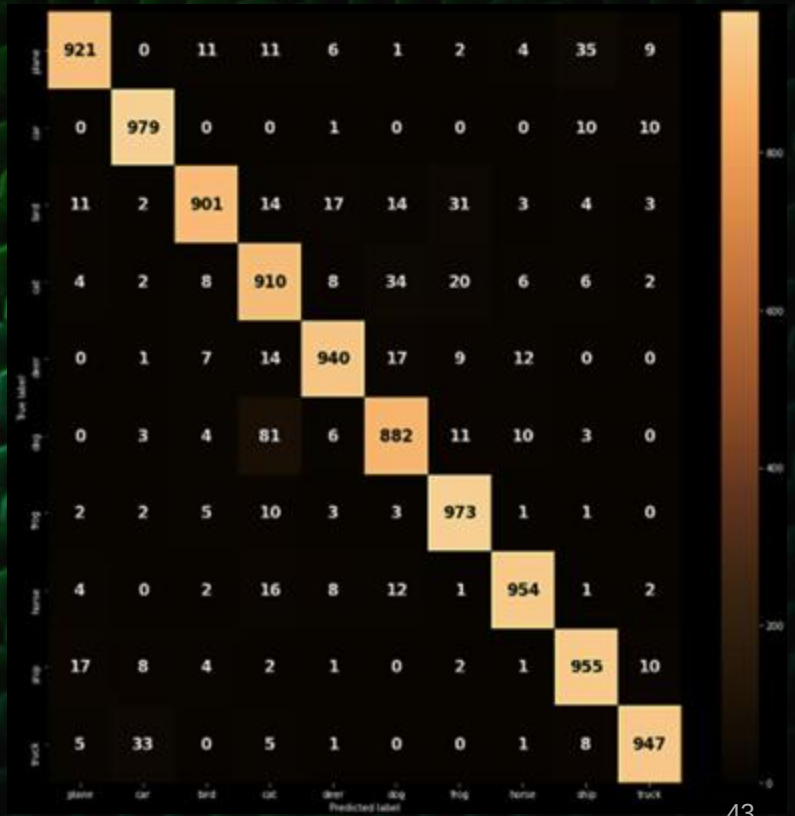


Reduction Cell



Results

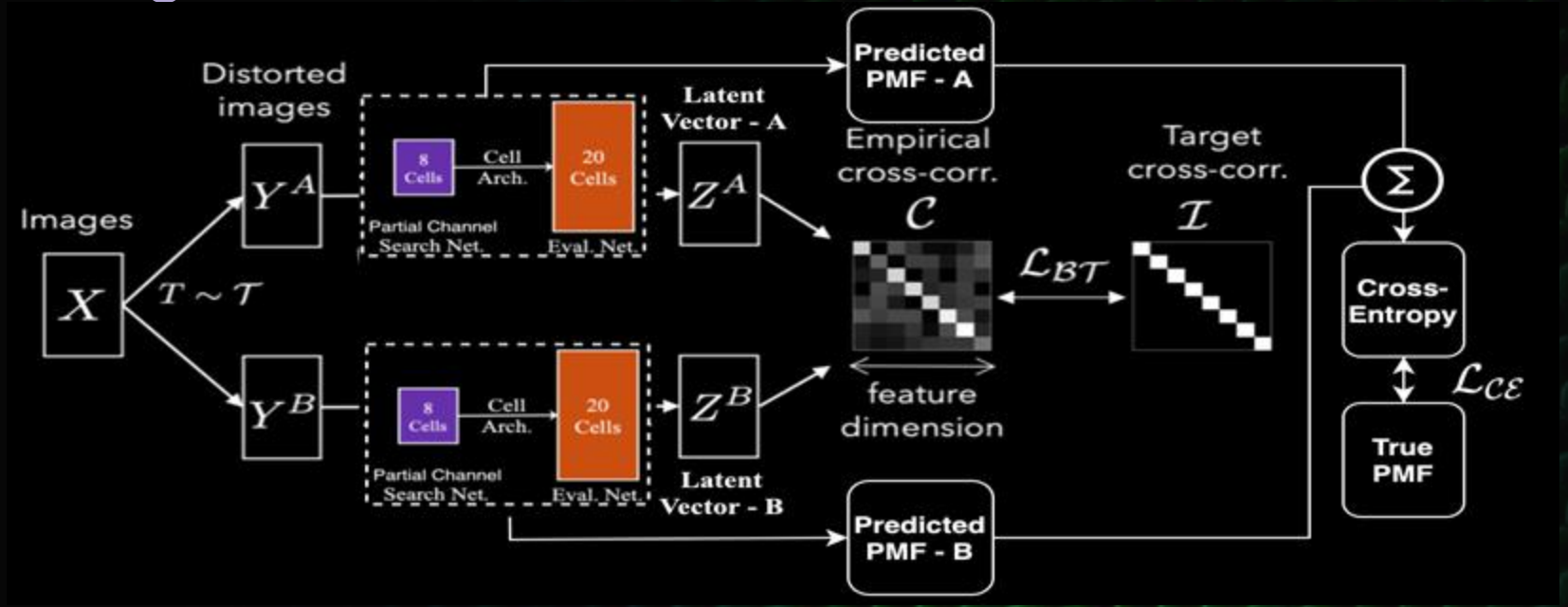
Class	precision	recall	f1-score	Accuracy
plane	0.955394	0.9210	0.937882	0.9878
car	0.950485	0.9790	0.964532	0.9928
bird	0.956476	0.9010	0.927909	0.9860
cat	0.856068	0.9100	0.882210	0.9757
deer	0.948537	0.9400	0.944249	0.9889
dog	0.915888	0.8820	0.898625	0.9801
frog	0.927550	0.9730	0.949732	0.9897
horse	0.961694	0.9540	0.957831	0.9916
ship	0.933529	0.9550	0.944142	0.9887
truck	0.963377	0.9470	0.955119	0.9911
all	0.936200	0.9362	0.936200	0.9362



FMNIST Transfer Train (proxy CIFAR-10)

Test Accuracy = 95.65

Stage 2 : SS PC DARTS

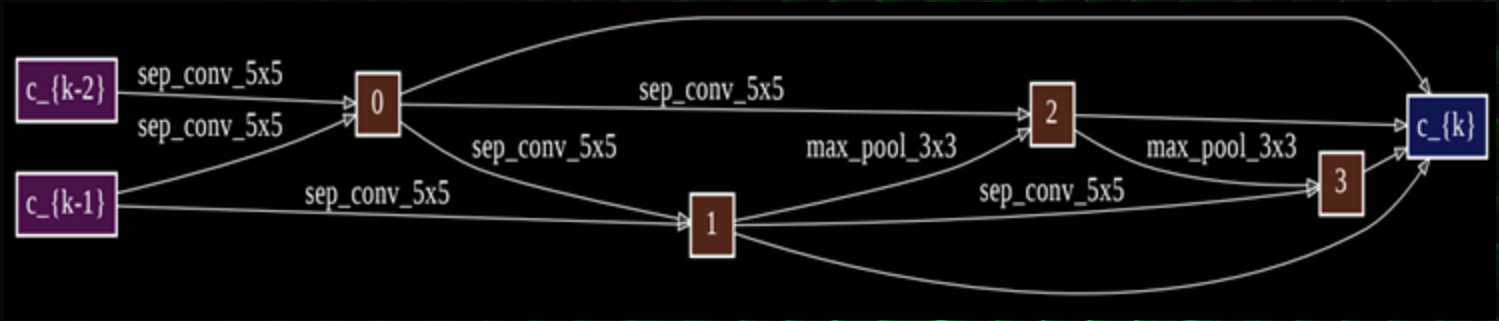


➔ Features are :

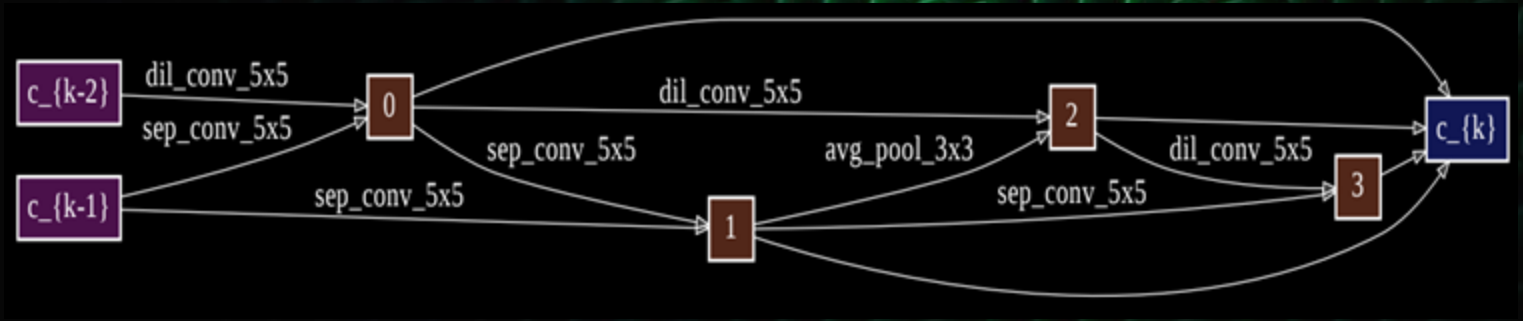
- Partial Connections based Search
- Mixture of Self Supervised & Supervised Loss for Search
- Supervised Loss for Classification

Search Stage Results → ss_factor = 0.75

Normal Cell

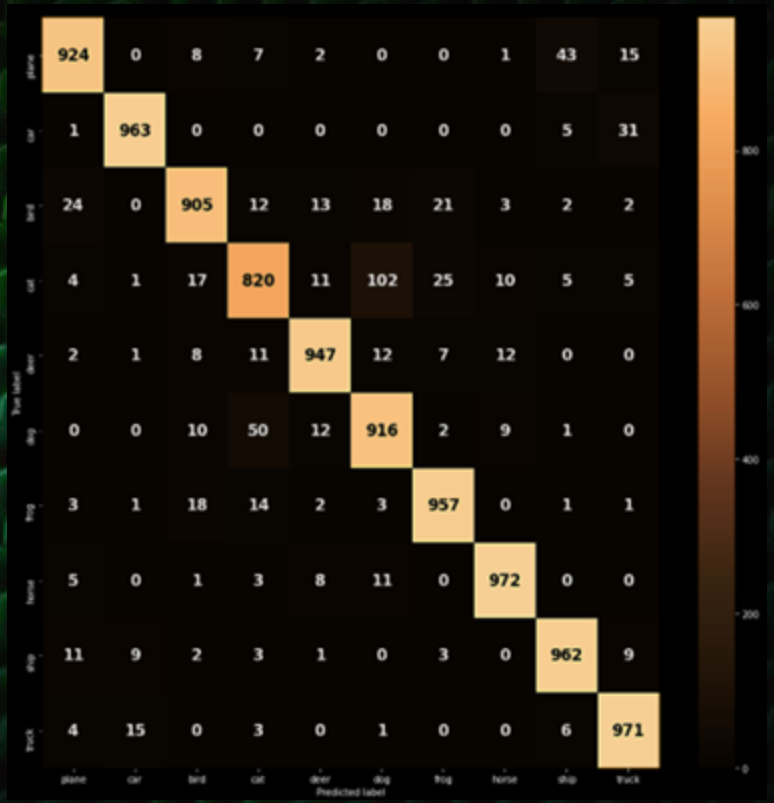


Reduction Cell



Results → ss_factor = 0.75

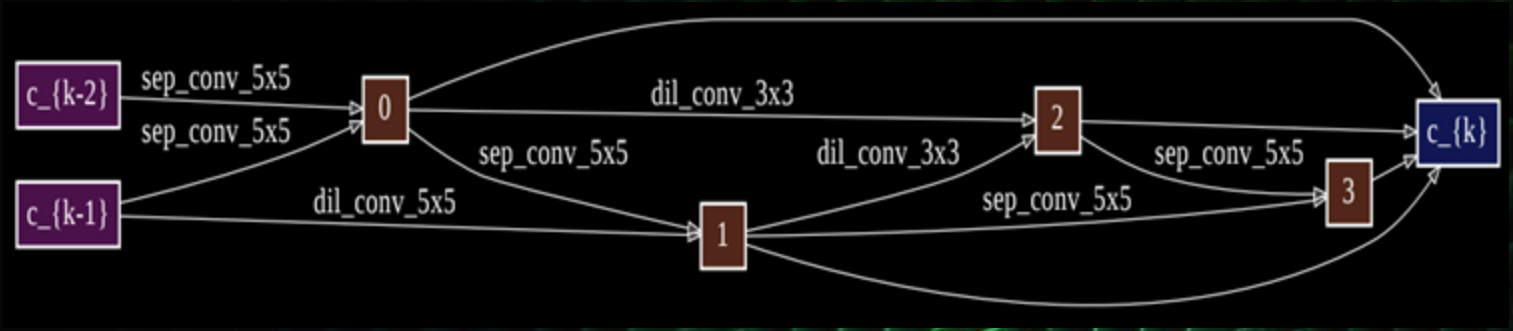
Class	precision	recall	f1-score	Accuracy
plane	0.944785	0.9240	0.934277	0.9870
car	0.972727	0.9630	0.967839	0.9936
bird	0.933953	0.9050	0.919248	0.9841
cat	0.888407	0.8200	0.852834	0.9717
deer	0.950803	0.9470	0.948898	0.9898
dog	0.861712	0.9160	0.888027	0.9769
frog	0.942857	0.9570	0.949876	0.9899
horse	0.965243	0.9720	0.968610	0.9937
ship	0.938537	0.9620	0.950123	0.9899
truck	0.939072	0.9710	0.954769	0.9908
all	0.933700	0.9337	0.933700	0.9337



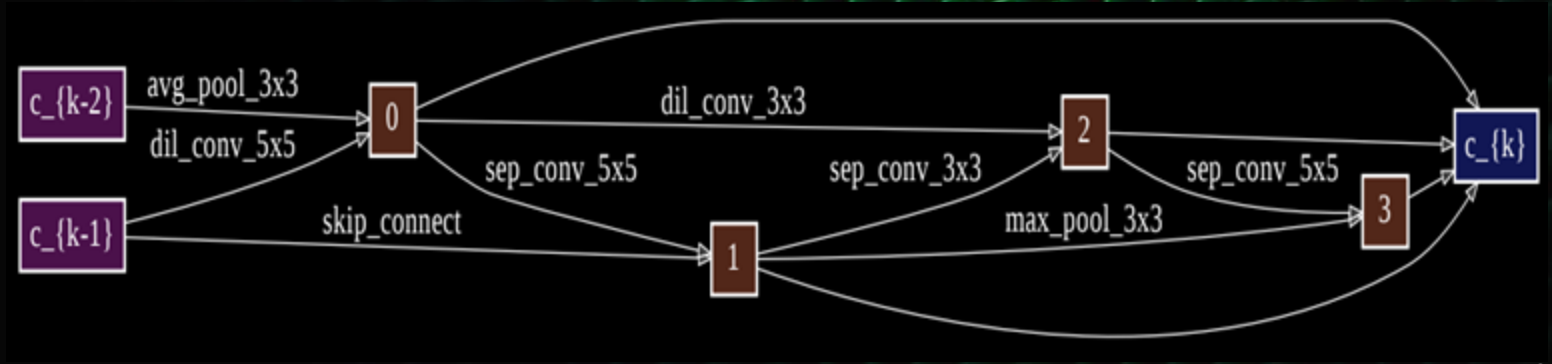
Prediction: plane
Probability: 81.8041

Search Stage Results → ss_factor = 1.00

Normal Cell

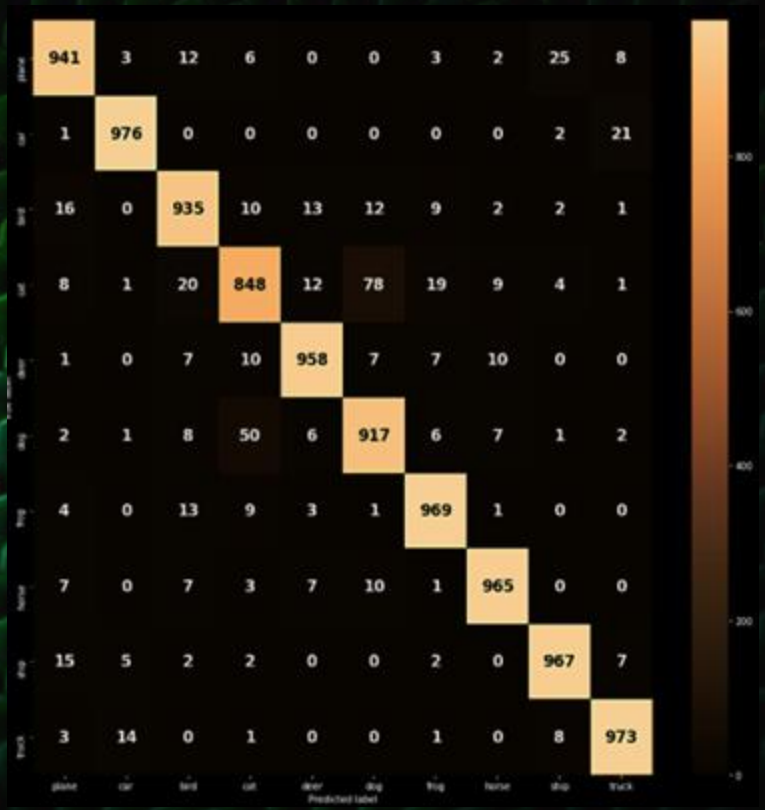


Reduction Cell



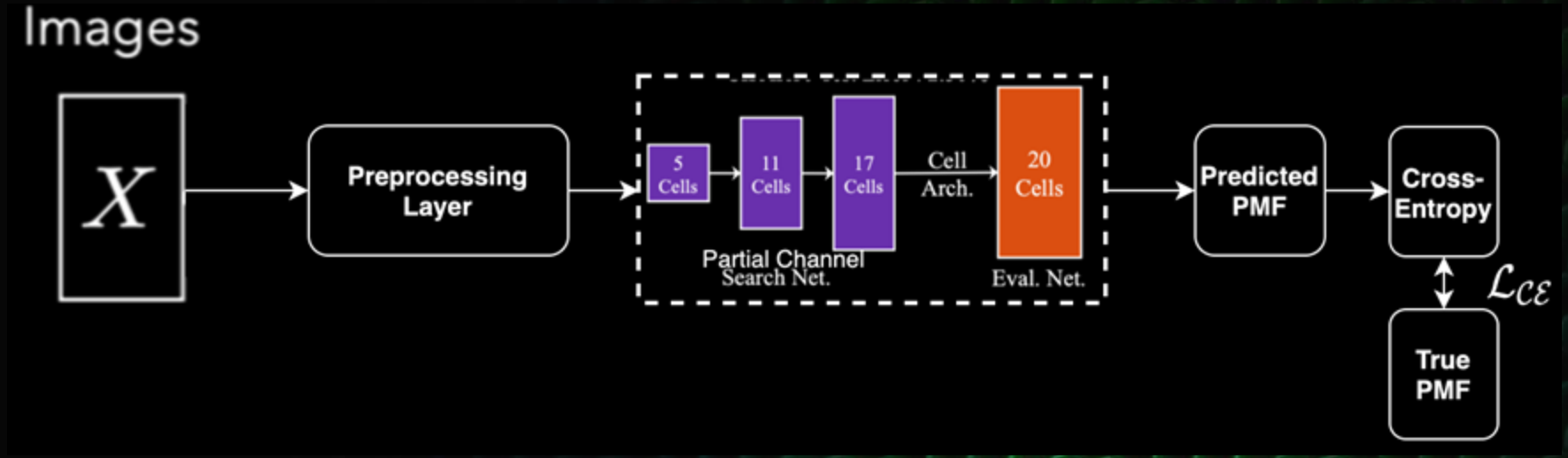
Results → ss_factor = 1.00

Class	precision	recall	f1-score	Accuracy
plane	0.942886	0.9410	0.941942	0.9884
car	0.976000	0.9760	0.976000	0.9952
bird	0.931275	0.9350	0.933134	0.9866
cat	0.903088	0.8480	0.874678	0.9757
deer	0.958959	0.9580	0.958479	0.9917
dog	0.894634	0.9170	0.905679	0.9809
frog	0.952802	0.9690	0.960833	0.9921
horse	0.968876	0.9650	0.966934	0.9934
ship	0.958375	0.9670	0.962668	0.9925
truck	0.960513	0.9730	0.966716	0.9933
all	0.944900	0.9449	0.944900	0.9449



Prediction: car
Probability: 99.0179

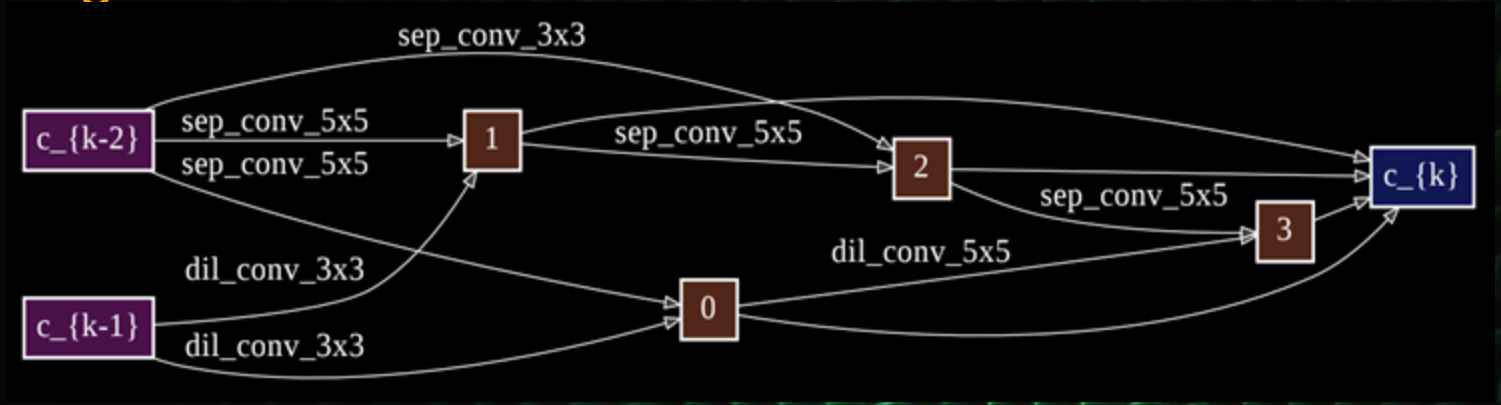
Stage 3 : Vanilla PCP Darts



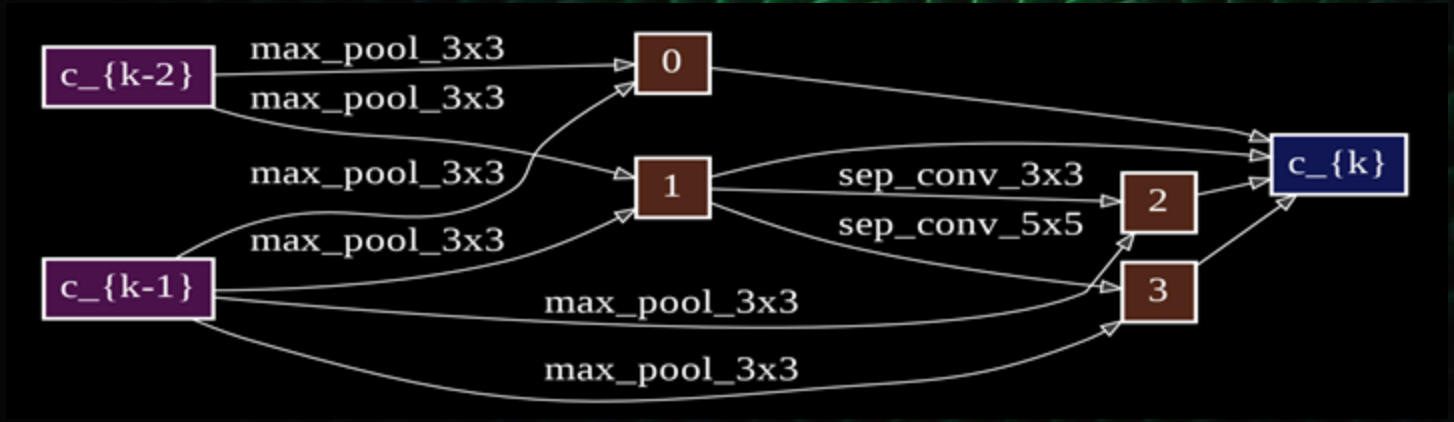
- ➔ Features are :
  - ◆ Progressive + Partial Connections based Search
  - ◆ Supervised Loss for Search and Classification

Search Stage Results

Normal Cell

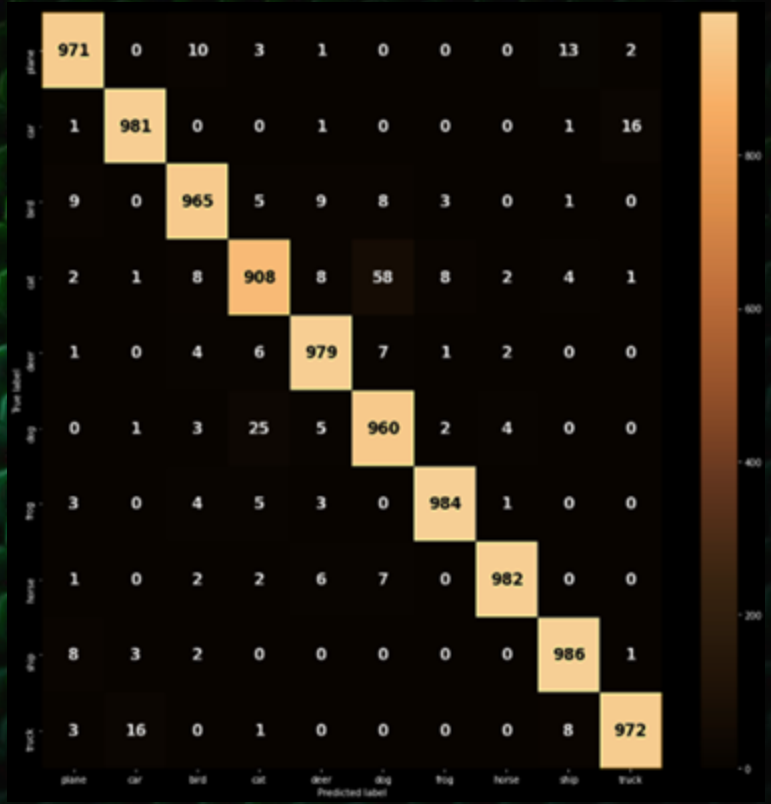


Reduction Cell



Results

Class	precision	recall	f1-score	Accuracy
plane	0.971972	0.9710	0.971486	0.9943
car	0.979042	0.9810	0.980020	0.9960
bird	0.966934	0.9650	0.965966	0.9932
cat	0.950785	0.9080	0.928900	0.9861
deer	0.967391	0.9790	0.973161	0.9946
dog	0.923077	0.9600	0.941176	0.9880
frog	0.985972	0.9840	0.984985	0.9970
horse	0.990918	0.9820	0.986439	0.9973
ship	0.973346	0.9860	0.979632	0.9959
truck	0.979839	0.9720	0.975904	0.9952
all	0.968800	0.9688	0.968800	0.9688



CIFAR-100 Transfer Train (proxy CIFAR-10)

↓

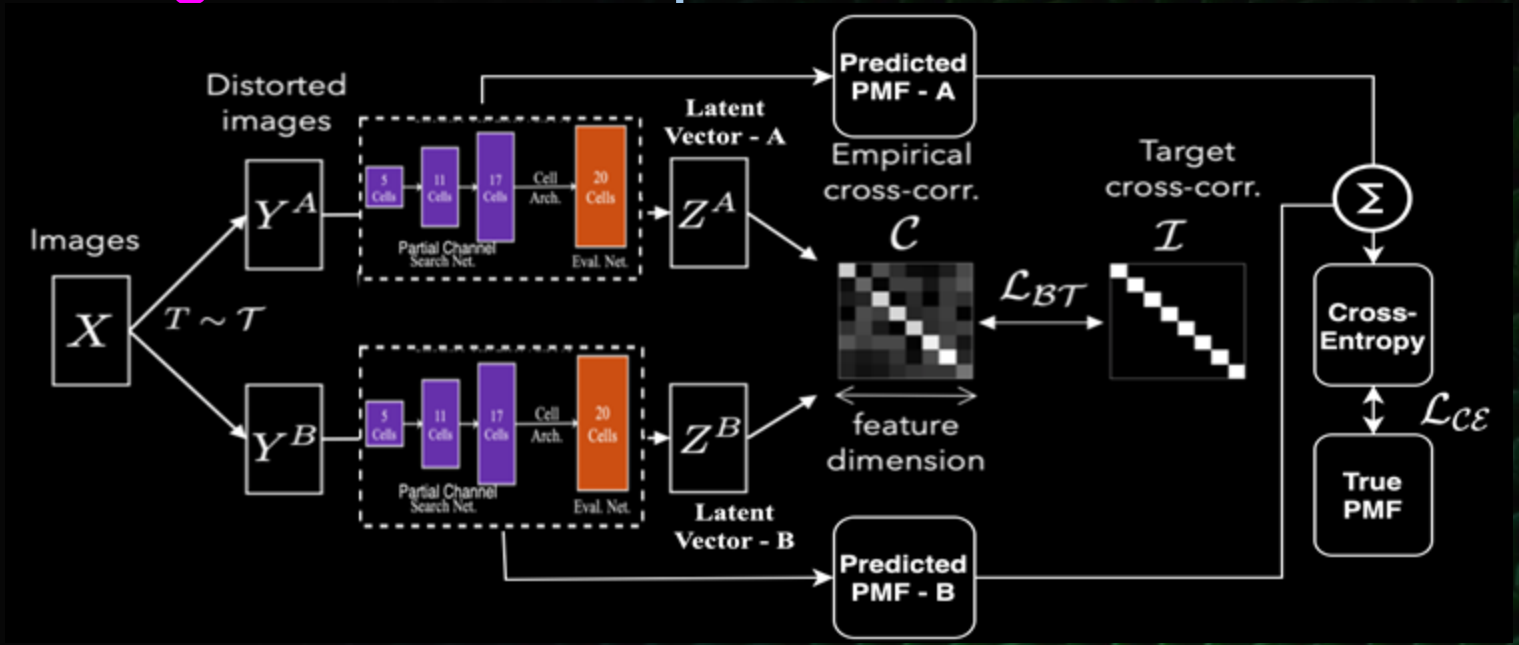
Test Accuracy = 81.92

FMNIST Transfer Train (proxy CIFAR-10)

↓

Test Accuracy = 95.97

Stage 4 : Self-Supervised PCP Darts

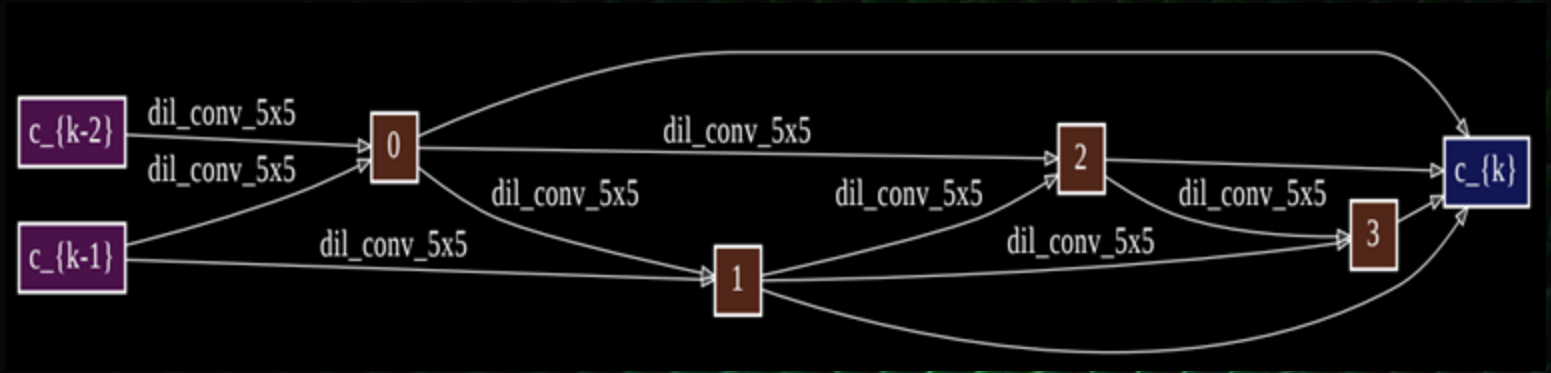


➔ Features are :

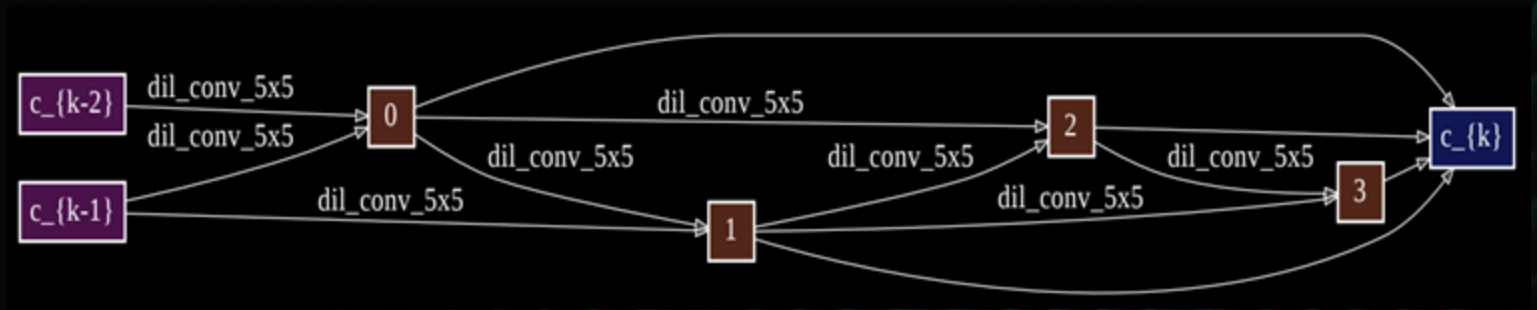
- Progressive + Partial Connections based Search
- Mixture of Self Supervised & Supervised Loss for Search
- Supervised Loss for Classification

Search Stage Results → ss_factor = 0.75

Normal Cell

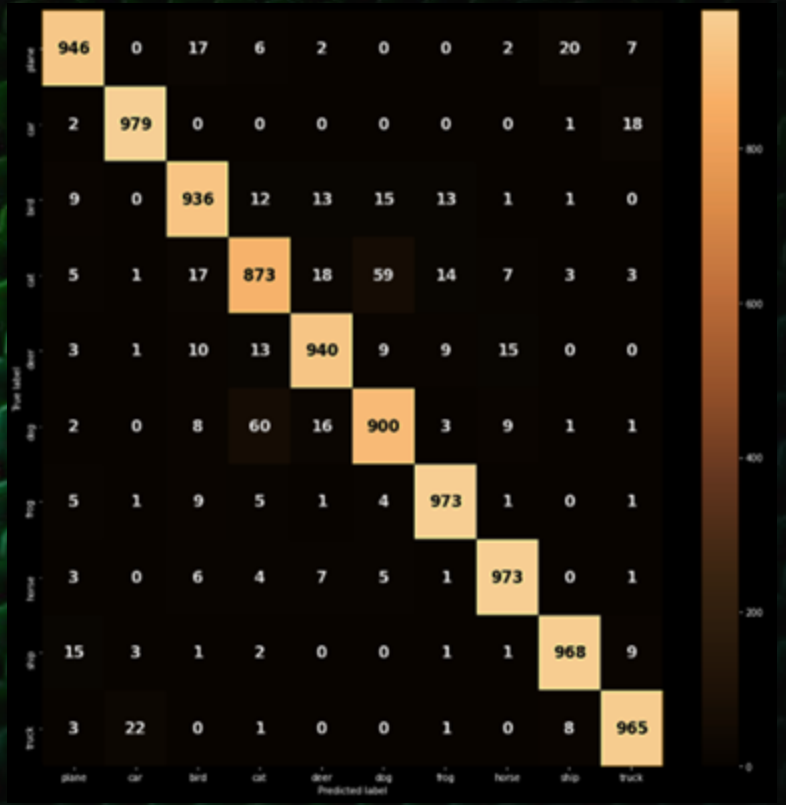


Reduction Cell



Results → ss_factor = 0.75

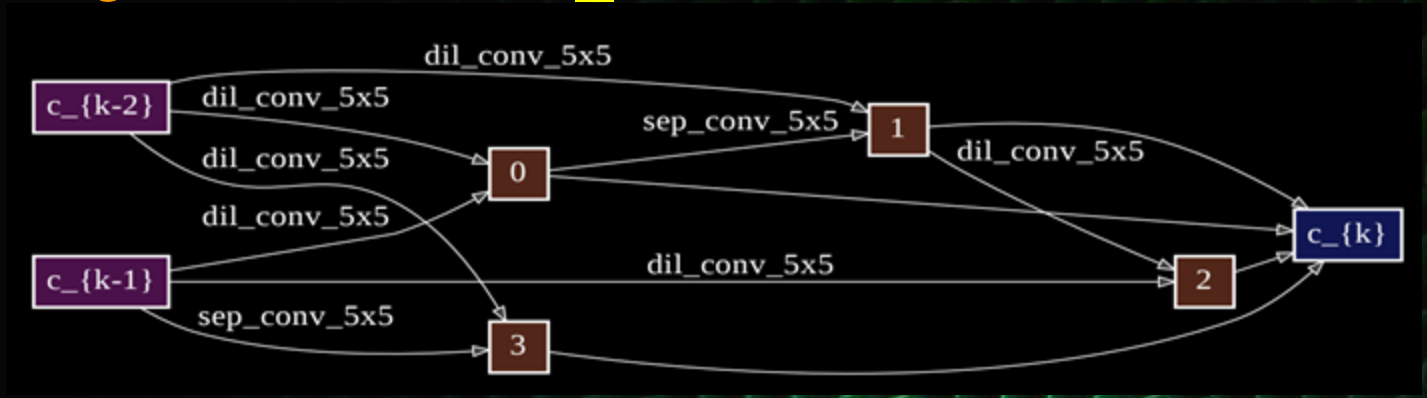
Class	precision	recall	f1-score	Accuracy
plane	0.952669	0.9460	0.949323	0.9899
car	0.972195	0.9790	0.975585	0.9951
bird	0.932271	0.9360	0.934132	0.9868
cat	0.894467	0.8730	0.883603	0.9770
deer	0.942828	0.9400	0.941412	0.9883
dog	0.907258	0.9000	0.903614	0.9808
frog	0.958621	0.9730	0.965757	0.9931
horse	0.964321	0.9730	0.968641	0.9937
ship	0.966068	0.9680	0.967033	0.9934
truck	0.960199	0.9650	0.962594	0.9925
all	0.945300	0.9453	0.945300	0.9453



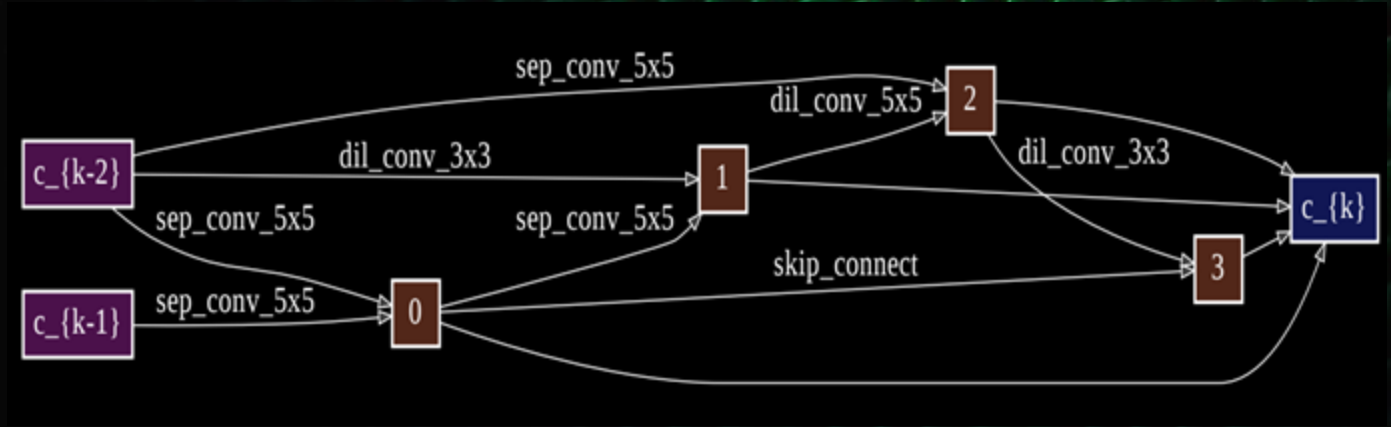

Prediction: ship
Probability: 99.9271

Search Stage Results → ss_factor = 1.00

Normal Cell

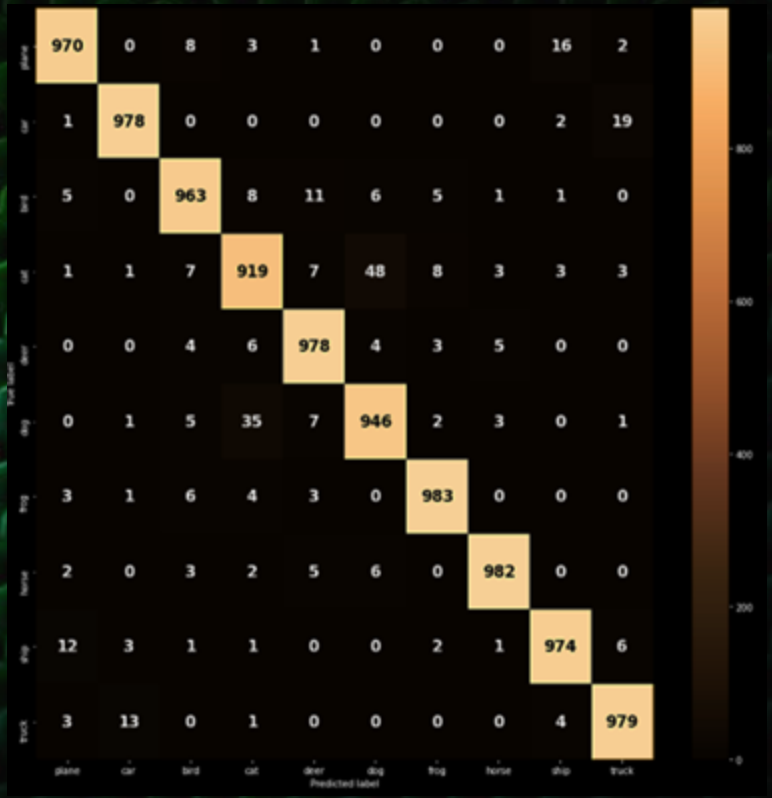


Reduction Cell



Results → ss_factor = 1.00

Class	precision	recall	f1-score	Accuracy
plane	0.972919	0.9700	0.971457	0.9943
car	0.980943	0.9780	0.979469	0.9959
bird	0.965898	0.9630	0.964447	0.9929
cat	0.938713	0.9190	0.928752	0.9859
deer	0.966403	0.9780	0.972167	0.9944
dog	0.936634	0.9460	0.941294	0.9882
frog	0.980060	0.9830	0.981528	0.9963
horse	0.986935	0.9820	0.984461	0.9969
ship	0.974000	0.9740	0.974000	0.9948
truck	0.969307	0.9790	0.974129	0.9948
all	0.967200	0.9672	0.967200	0.9672



CIFAR-100 Transfer Train (proxy CIFAR-10)

↓

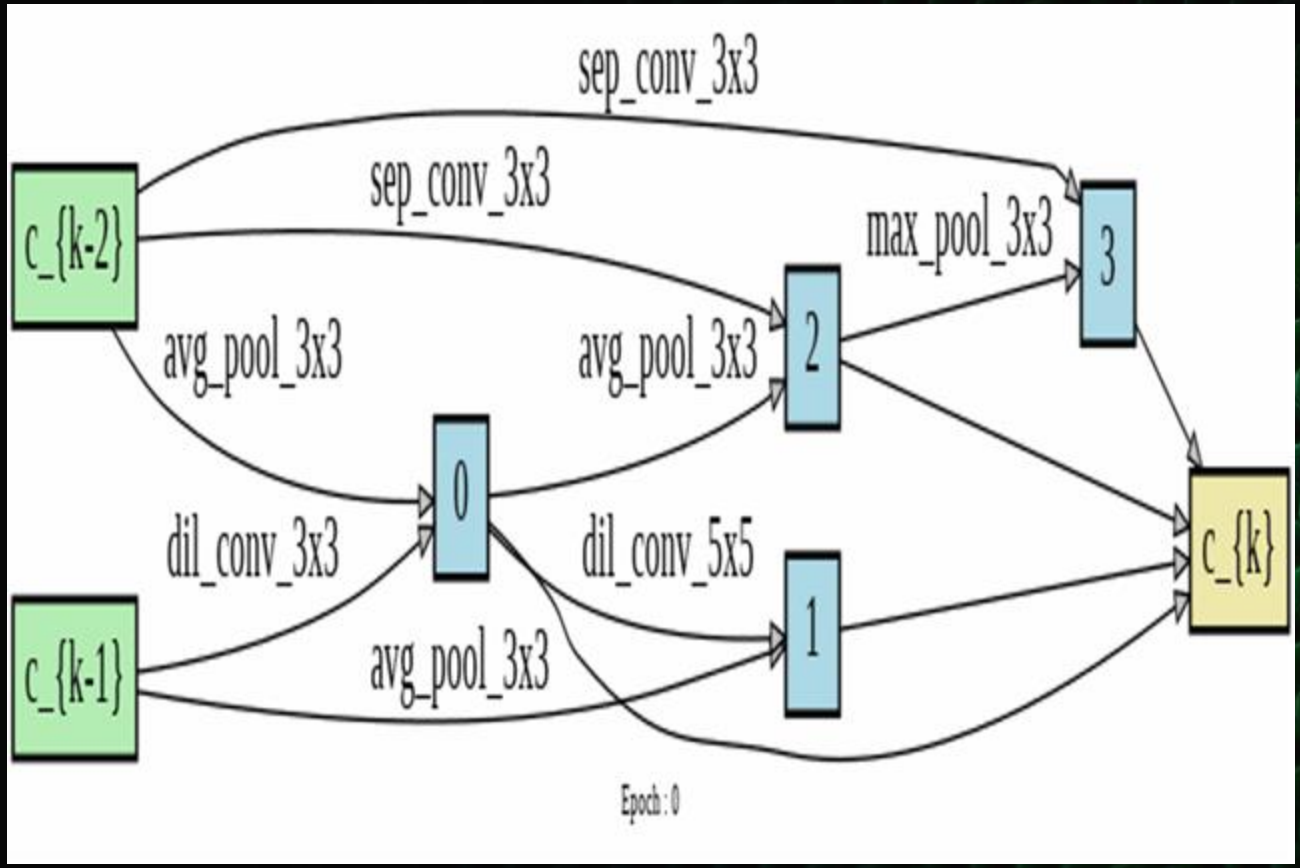
Test Accuracy = 74.65

FMNIST Transfer Train (proxy CIFAR-10)

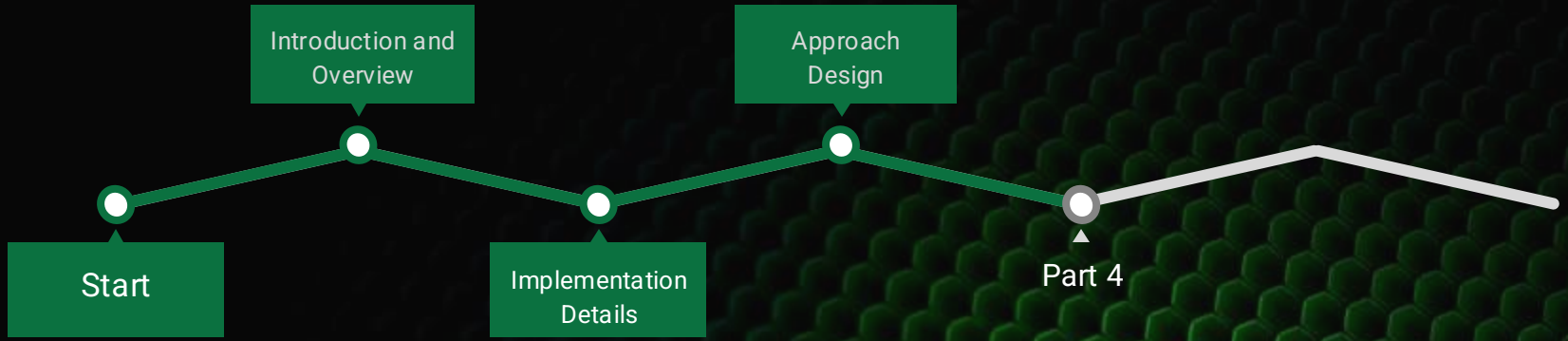
↓

Test Accuracy = 96.06

Neural Architecture Search in Action !



Evolution of Reduction Cell in Vanilla PC DARTS


Introduction and Overview
Approach Design
Start
Implementation Details
Part 4

Analysis

Comparison between various stages on CIFAR-10 Dataset

Stages	Description	SS Factor	Test top-1 Accuracy	Network Search Days	No. of Parameters
Stage-1	Vanilla PC Darts	0	93.62	0.1610	3.9 M
Stage-2	SS PC Darts	0.25	94.81	0.2223	4.6 M
		0.5	92.01	0.2158	1.8 M
		0.75	93.37	0.2921	4.6 M
		1	94.49	0.2314	4.7 M
Stage-3	Vanilla PCP Darts	0	96.88	0.1883	4.5 M
Stage-4	SS PCP Darts	0.25	96.09	0.2346	3.9 M
		0.5	94.07	0.1889	3.7 M
		0.75	94.53	0.3535	3.7 M
		1	96.72	0.2679	4.2 M

Time and Test Accuracy Comparison

- ➔ Time taken for Architecture Search:

  PC-DARTS $\lesssim$ PCP-DARTS $<$ P-DARTS

- ➔ PCP-DARTS takes much less time as compared to P-DARTS.
- ➔ PC-DARTS and PCP-DARTS take comparable time for architectural search

- ➔ Test Accuracy:

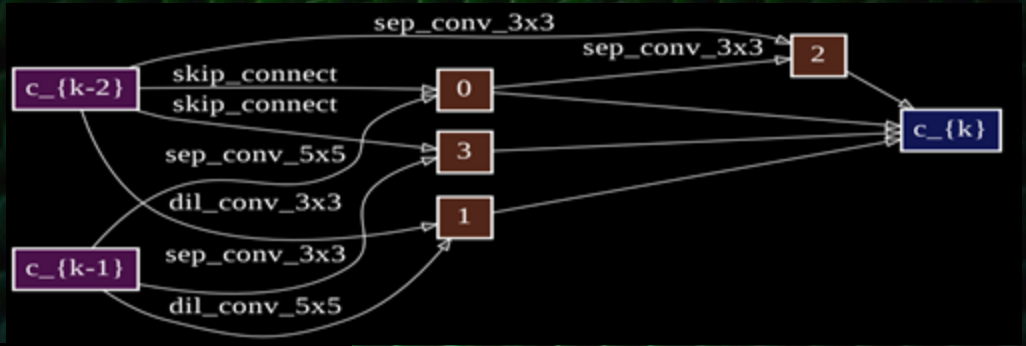
  PC-DARTS $<$ PCP-DARTS $\lesssim$ P-DARTS

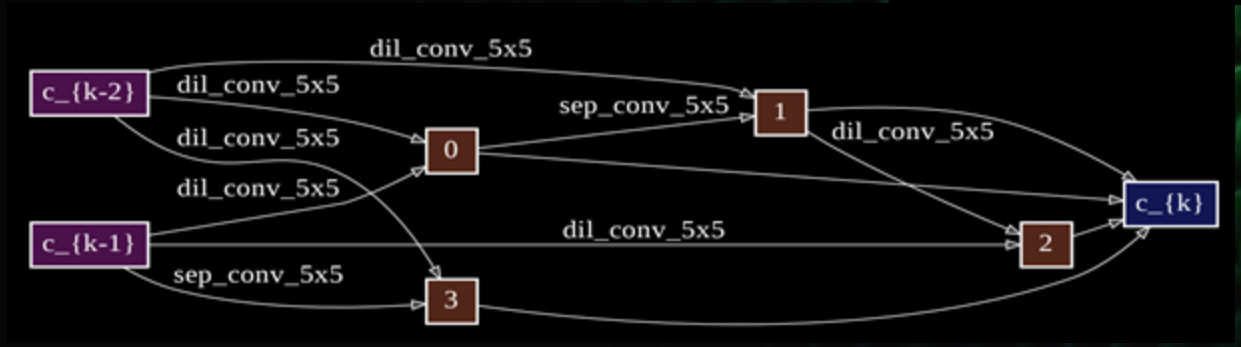
- ➔ PCP-DARTS shows better accuracy than PC-DARTS
- ➔ But PCP-DARTS and P-DARTS show almost similar test accuracy.
- ➔ PCP-DARTS has overall better accuracy and takes lesser time. It combines the best of both worlds

Architecture Comparison

The vanilla PCP-DARTS has less number of skip connections and more depth than vanilla PC-DARTS

Vanilla PC-DARTS Normal Cell →



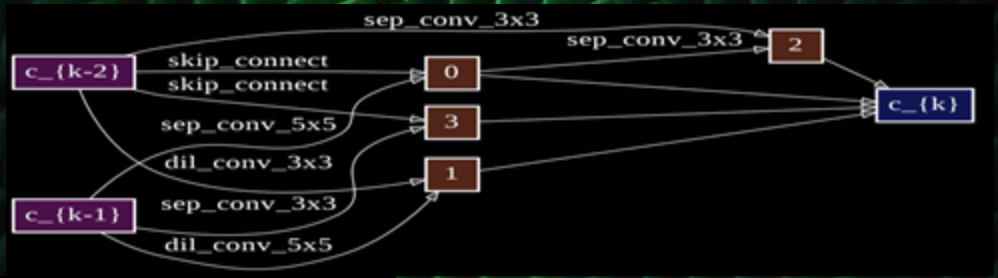


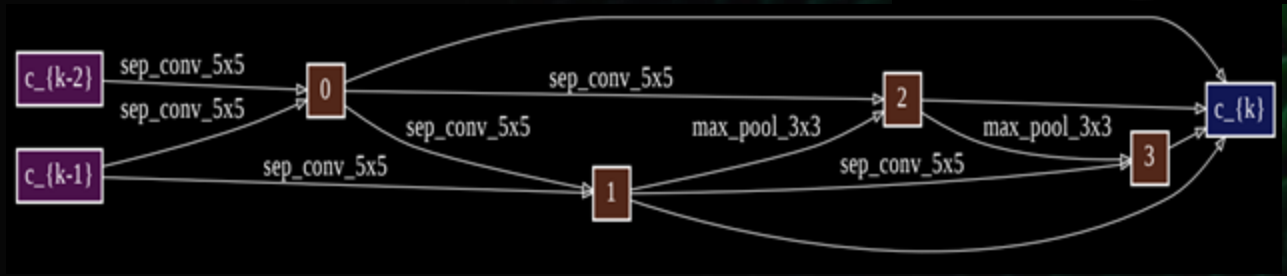
← Vanilla PCP-DARTS Normal Cell

Does Self-Supervised Learning aid in increasing depth??

- We observed from the normal and reduction cells that Self-Supervised Architecture Search leads to the network to have more depth.
- Possibly diversity, richness and quality of features improve with depth, which is ensured by using Barlow Twin Loss.

Vanilla PC-DARTS Normal Cell →



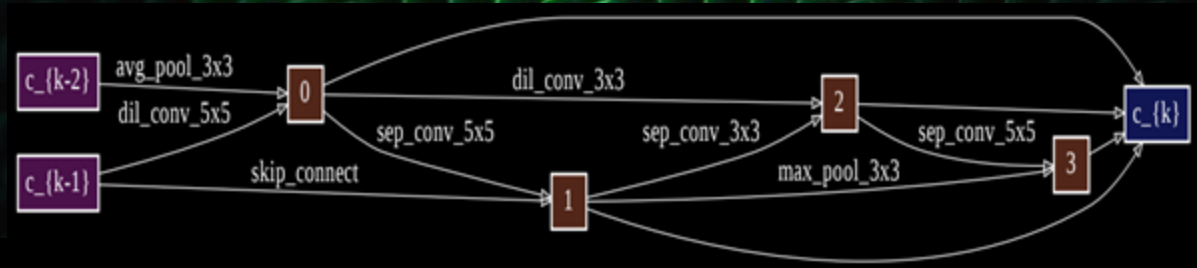


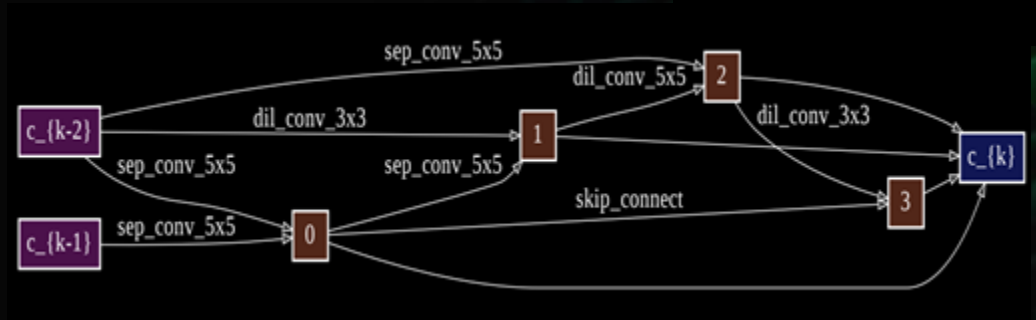
← SS(0.75) PC-DARTS Normal Cell

PC-DARTS prefers parameter-less operations

- Parameter less operations like pooling and skip-connections are preferred in PC-DARTS as compared to PCP-DARTS.
- For transfer training on harder (classification-wise) datasets like ImageNet, there is a chance that PC-DARTS arch. might perform inferiorly.

SS(1) PC-DARTS Reduction Cell →

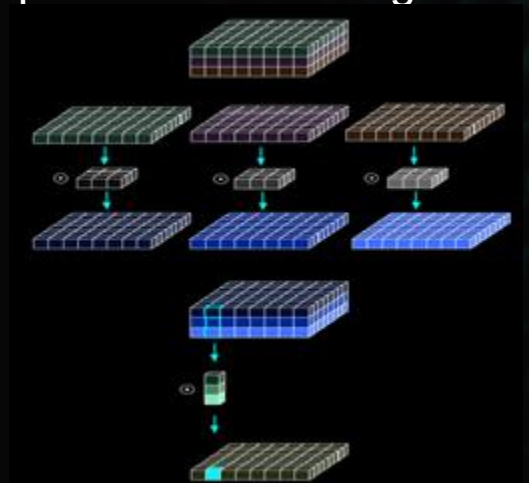


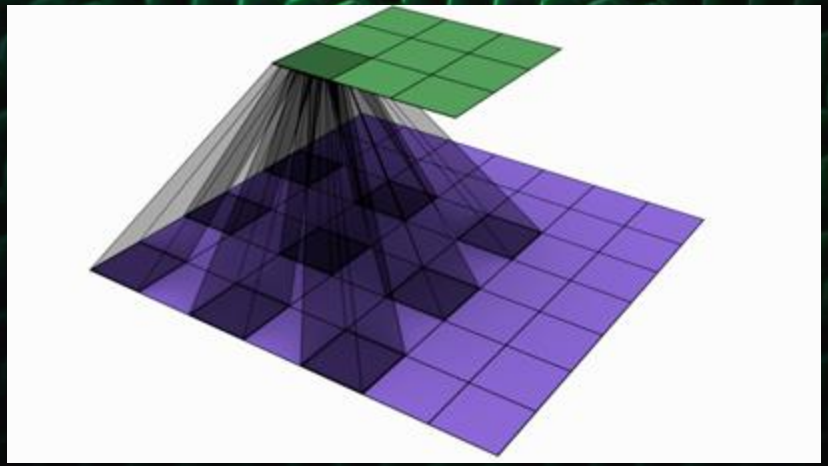


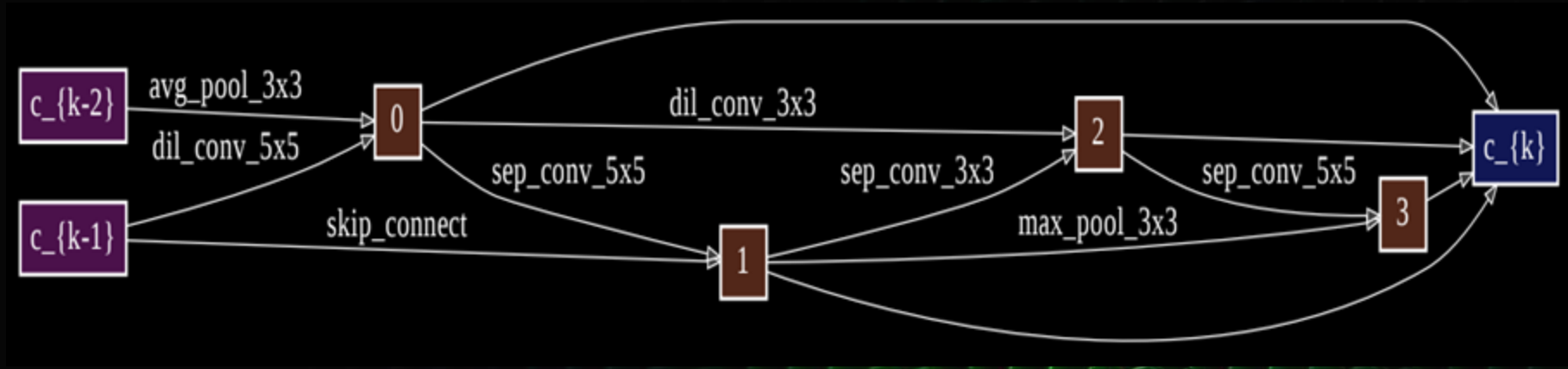
← SS(1) PCP-DARTS Reduction Cell

PCP-DARTS has more representational power than PC-DARTS

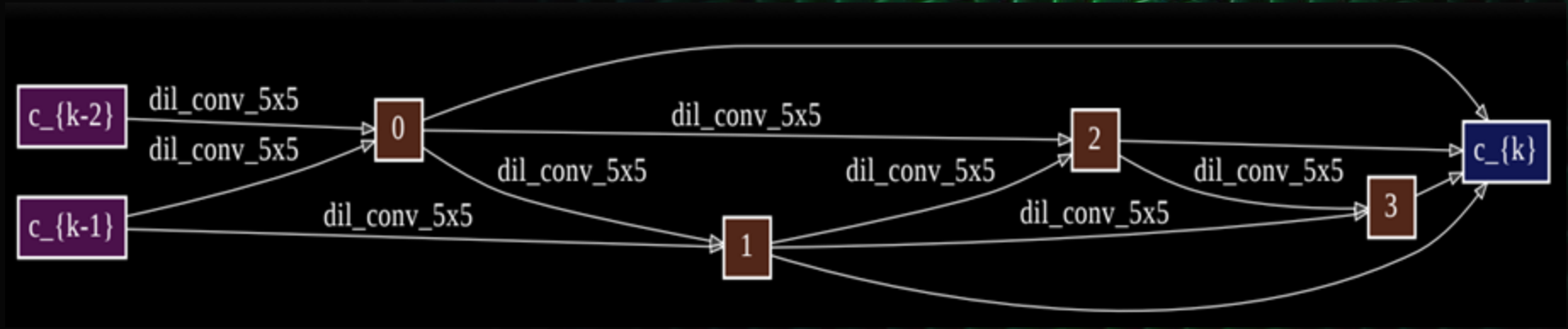
- ➔ Separable Convolution + Pooling is a repeated pattern seen in PC DARTS ⇒ Requires two operations for having a larger receptive field.
- ➔ It is observed in PC-DARTS that Pooling is mostly preceded by a separable convolution
- ➔ In PCP-DARTS Dilated Convolution are commonly seen ⇒ Here only one operation has a larger receptive field





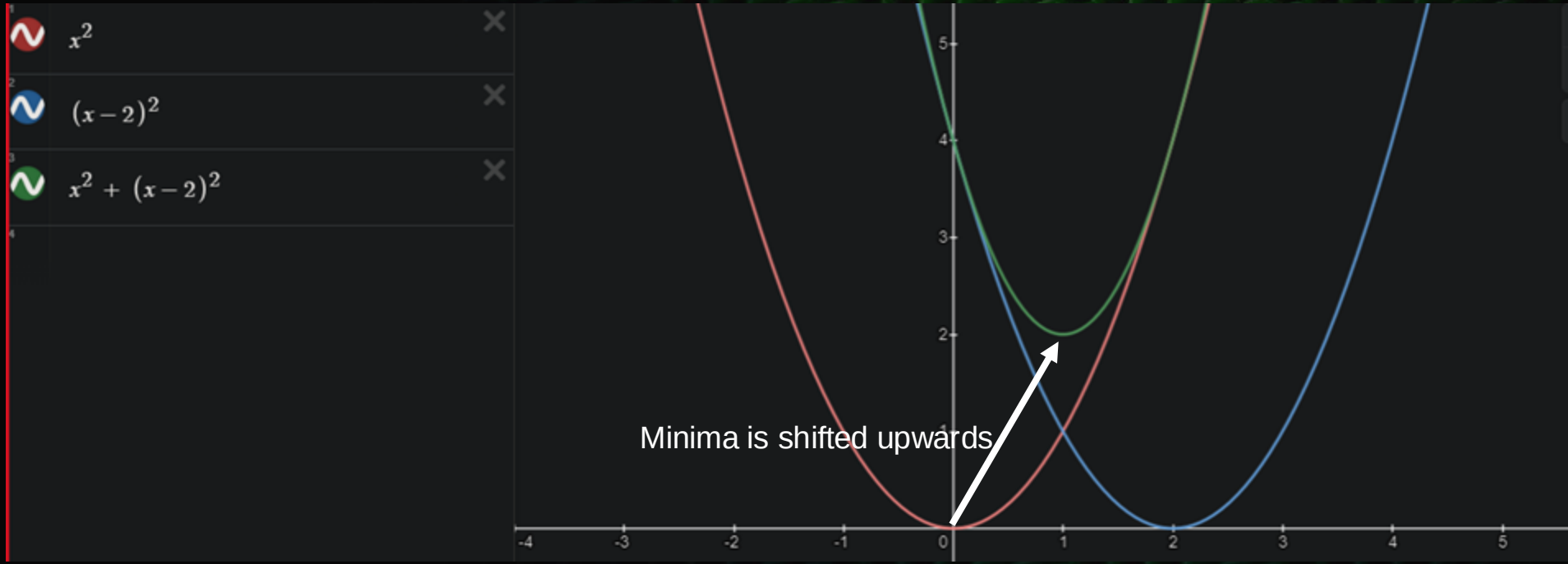


SS(1) PC-DARTS Reduction Cell



SS(0.75) PCP-DARTS Reduction Cell

Mixture of two objective functions – Barlow Twin Loss + Cross Entropy Loss – might force the network to juggle between two (possibly) different minimas, leading to slightly less performance.



Adversarial Attack Method : FGSM

- ➔ Generating adversarial test examples to fool Deep Learning models : Provides insight into the robustness of the approaches
- ➔ Adversarial Example : a maliciously designed input which is perceptually indistinguishable from original input but is misclassified by the model
- ➔ Fast Gradient Sign Method : Computationally Efficient Way of generating adv. examples

$$X^{adv} = X + \epsilon \, \text{sign}\left(\nabla_X J\left(X, Y_{\text{true}}\right)\right.$$

where

$X =$ original (clean) input

$X_{adv} =$ adversarial input

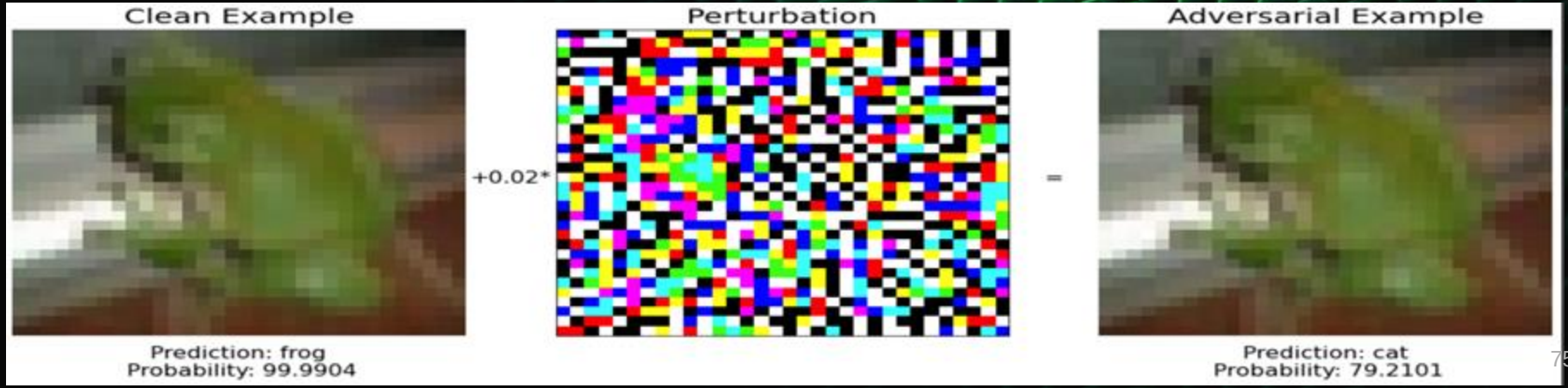
$\epsilon =$ magnitude of adversarial perturbation

$\nabla_X J\left(X, Y_{true}\right) =$ gradient of loss function w.r.t to input (X)

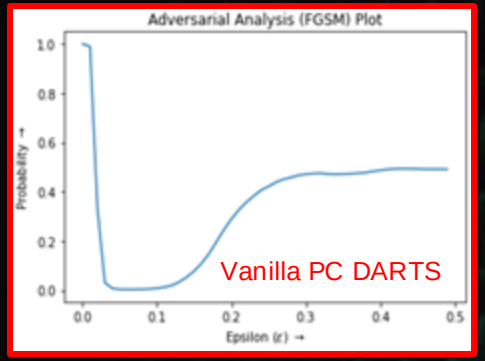
Like Gradient Ascent move in direction of most likely misclassification !

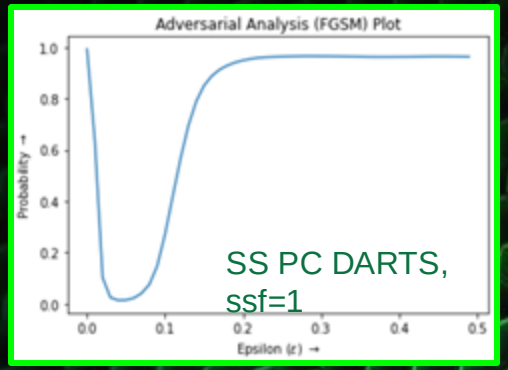
Adversarial Attack Analysis : Graphical Results

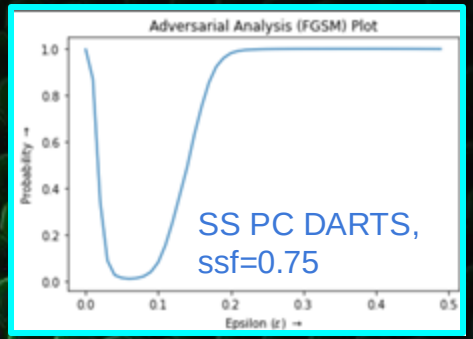
Adversarial Attack Analysis



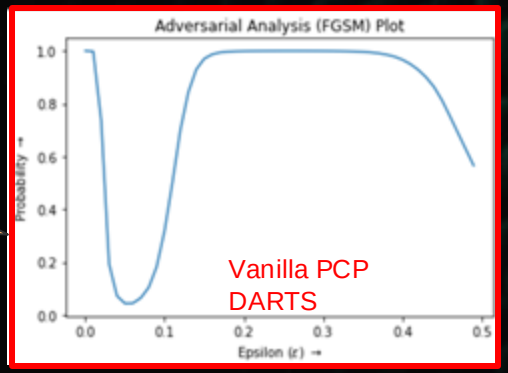
Adversarial Attack Analysis : Probability vs ε

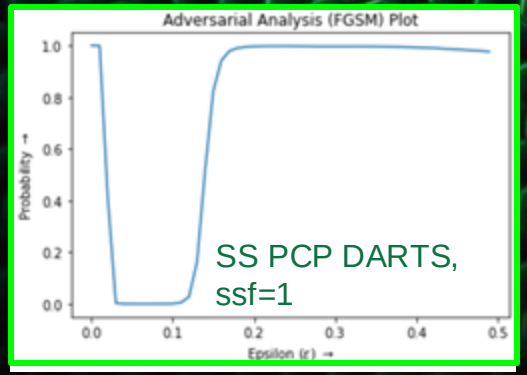


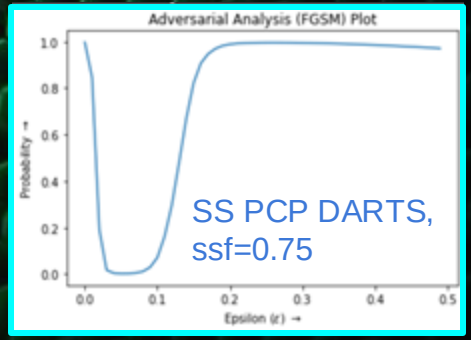




Interesting Observations







Adversarial Attack Analysis : Probability vs ε

➔ In the examples considered, all architectures found by each method predict the true class. (Here, Frog)

➔ Self-Supervision is incorporated in the objective function of architectural search → for most of the span of epsilon values considered it shows higher probability for the correct class

↓

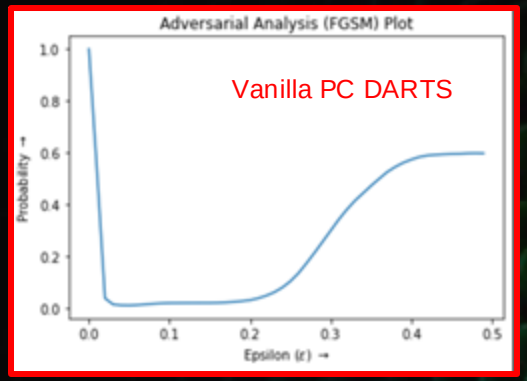
some amount of Barlow Twin SS, is making the found architecture slightly robust to adversarial attacks like FGSM

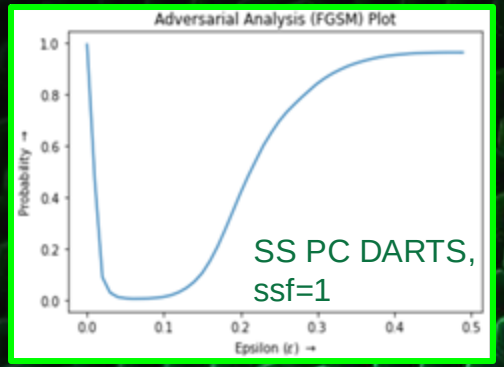
➔ Dip in the the plot, of prob vs epsilon, for a small range of epsilon values

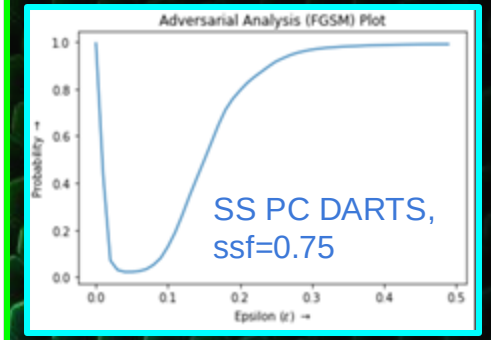
↓

Because of the non-task specific loss function, the SS loss function may contain multiple minima for the same class as compared to vanilla (Fully-supervised) loss function.

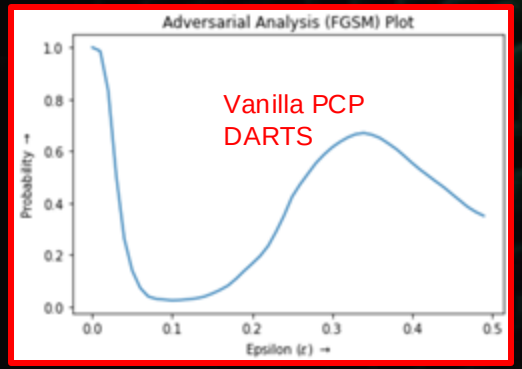
Adversarial Attack Analysis : Probability vs ε

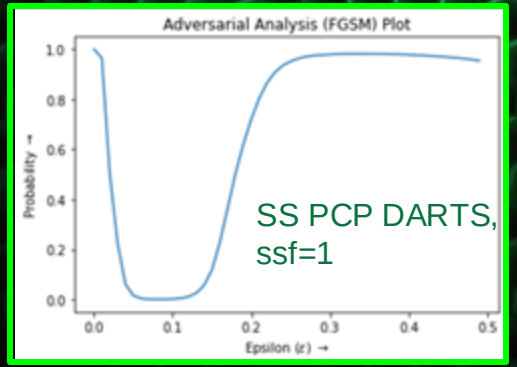


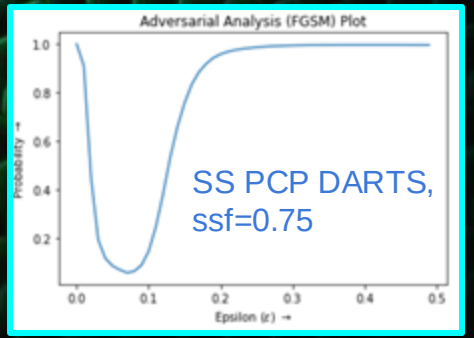




Interesting Observations





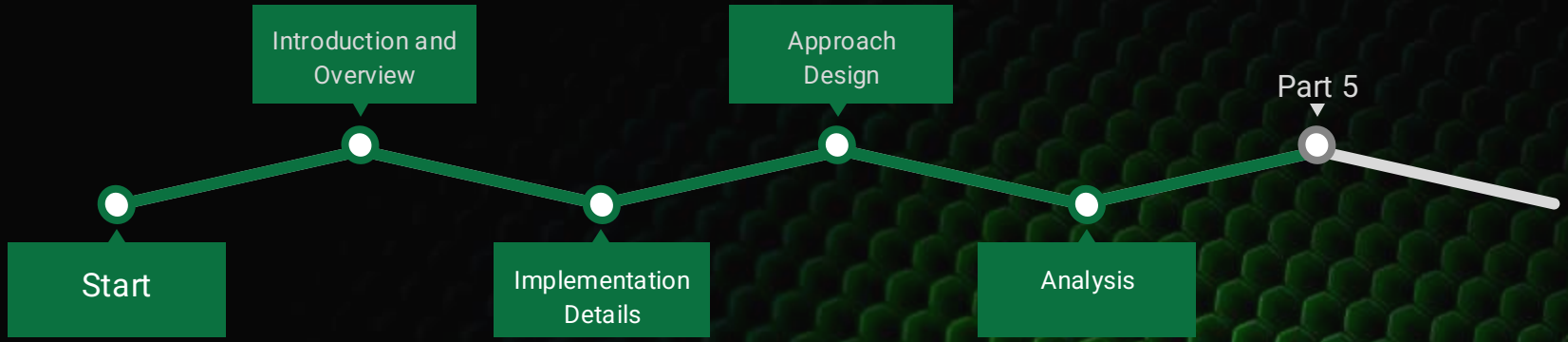


Adversarial Attack Analysis : Probability vs ε

➔ In the example considered, all architectures found by each method predict the true class. (Here, Frog)

➔ Self-Supervision is forcing the algorithm to choose operations that extract higher quality, differentiating features among classes, which is indicated in the Adversarial Attack plots.

- Higher probability predictions for the chosen predicted class
- Decorrelation of decision regions to some extent because of SS learning

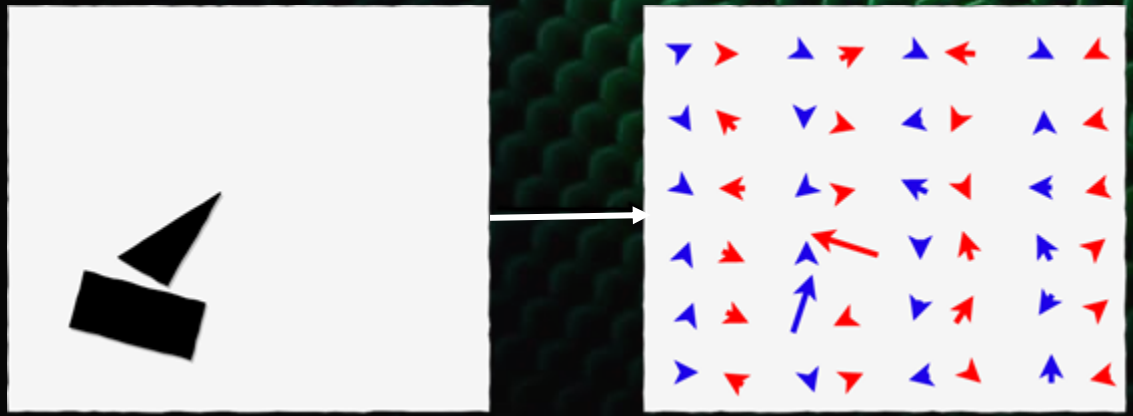


Conclusion and Future Work

Car, p = 0.94
Conclusion, p = 0.99999

Capsule Networks

- ➔ Capsule Networks are a promising domain in Deep Learning. They have already surpassed CNN's accuracy over small datasets like MNIST.
- ➔ We tried to include Capsule Networks with three different routings in our NAS operation space also.
- ➔ Some of the problems we faced are listed below
  - ◆ Capsule Network are computationally very expensive and doing a NAS over Capsule Network usually takes several GPU-days
  - ◆ The PC-DARTS was eliminating the Capsule Network Operations after the first few iterations



Future Work

- ➔ Search stage and Train Stage can be run for more epochs
- ➔ Making the found architectures and learnt weights more robust to adversarial attacks (for all epsilon values in case of FGSM)
- ➔ Make the capsule layers less computational intensive so that they can be incorporated in the operation search space.
- ➔ Explore and Analyse the incorporation of self-supervision for classification
- ➔ Explore and analyse transfer performance on other tasks (Object Detection etc.)
- ➔ Explore and analyse transfer train on harder datasets (ImageNet etc.)

References

- Chen, Xin, et al. "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation." *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.

- Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." *arXiv preprint arXiv:1806.09055* (2018).

- Xu, Yuhui, et al. "PC-DARTS: Partial channel connections for memory-efficient architecture search." *arXiv preprint arXiv:1907.05737* (2019).

- Timofeev, Aleksandr, Grigorios G. Chrysos, and Volkan Cevher. "Self-Supervised Neural Architecture Search for Imbalanced Datasets." *arXiv preprint arXiv:2109.08580*(2021).

- Pham, Hieu, et al. "Efficient neural architecture search via parameters sharing." *International Conference on Machine Learning*. PMLR, 2018.

- Marchisio, Alberto, et al. "NASCaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks." *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020.

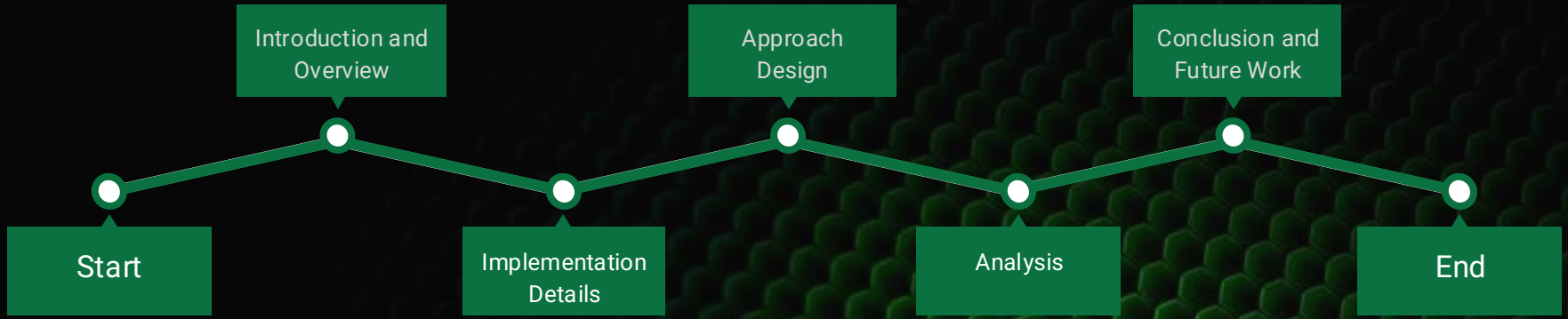
- Zbontar, Jure, et al. "Barlow twins: Self-supervised learning via redundancy reduction." *arXiv preprint arXiv:2103.03230*(2021).

- Zhao, Yiyang, et al. "Few-shot neural architecture search." *International Conference on Machine Learning*. PMLR, 2021.

- Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

Authors

- Asish kumar Mishra (SC18B074)
- Sri Aditya Deevi (SC18B080)


Introduction and Overview
Approach Design
Conclusion and Future Work
Start
Implementation Details
Analysis
End

Thank You !